TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Výpočty elektrické rozvodné sítě v prostředí webu

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Autor práce: **Bc. Martin Bureš**
Vedoucí práce: Ing. Jana Vitvarová, PhD.

Liberec 2016

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Web based calculations of power distribution network

Diploma thesis

Study programme: N2612 – Electrotechnology and informatics
Study branch: 1802T007 – Information technology

Author: **Bc. Martin Bureš**
Supervisor: Ing. Jana Vitvarová, PhD.

Liberec 2016

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2015/2016**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Martin Bureš**
Osobní číslo:	**M14000152**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Informační technologie**
Název tématu:	**Výpočty elektrické rozvodné sítě v prostředí webu**
Zadávající katedra:	**Ústav mechatroniky a technické informatiky**

Zásady pro vypracování:

1. Seznamte se s typy výpočtů prováděných nad elektrickou rozvodnou sítí. Pro účely implementace se zaměřte na výpočet ustáleného chodu sítě.
2. Zvolte vhodné webové technologie a metodiku testování softwaru.
3. Vytvořte systém, který bude sloužit pro editaci zvolené sítě, a bude umožňovat nad ní provádět zvolený typ výpočtu.
4. Otestujte systém a ověřte jeho funkčnost na reálném schématu sítě.
5. Zhodnoťte použití zvolených postupů a technologií.

Rozsah grafických prací: **dle potřeby dokumentace**

Rozsah pracovní zprávy: **cca 40–50 stran**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] **Kučera R.: Numerické metody. VŠB - Technická univerzita, Ostrava, 2006**

[2] **Čermák L., Hlavička R.: Numerické metody. Akademické nakladatelství CERM, Brno, 2008**

[3] **Richardson L., Ruby S.: RESTful web services. O'Reilly, 2007**

Vedoucí diplomové práce: **Ing. Jana Vitvarová, Ph.D.**
Ústav mechatroniky a technické informatiky

Datum zadání diplomové práce: **10. října 2015**

Termín odevzdání diplomové práce: **16. května 2016**

prof. Ing. Václav Kopecký, CSc.
děkan

L.S.

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 10. října 2015

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 12.5.2016

Podpis: Bart Nachi

Abstrakt

Diplomová práce popisuje návrh a vývoj softwarové platformy určené k řešení vybraných vědeckých výpočtů nad elektrizační soustavou v prostředí webu s využitím současných webových technologií. Jsou zde řešeny aktuální možnosti tvorby platformy z pohledu náročných výpočetních úloh, způsobů ukládání a sdílení dat napříč platformou, grafické vizualizace dat a propojení s externími systémy. Důraz je kladen především na snadné a modulární využití služeb platformy v případě reálného nasazení.

Klíčová slova: web, webové služby, platforma, elektrizační soustava, ustálený chod

Abstract

Diploma thesis describes the development of a software platform designed to solve selected scientific calculations on a power grid in a web environment using modern Web technologies. Current possibilities for the platform development are examined from the perspective of demanding computational tasks, ways of storing and sharing data across the platform, graphical data visualization and integration with external systems. The focus is primarily on an easy and modular use of the platform services in case of a real deployment.

Keywords: web, web services, platform, electricity grid, steady state

Poděkování

Děkuji Ing. Janě Vitvarové, Ph.D. za vedení mé diplomové práce.

Dále děkuji pracovníkům oddělení Provozu řídících systémů Východ – Hradec Králové ze společnosti ČEZ Distribuce a.s. za konzultace a možnost ověření vytvořených aplikací v provozních podmínkách.

Obsah

Seznam zkratek

ES	elektrizační soustava
GUI	graphical user interface, grafické uživatelské rozhraní
UI	user interface, uživatelské rozhraní
SaaS	software as a service, software jako služba
SPA	single page application, jednostránková aplikace
ER	entity relation diagram, diagram vztahů mezi entitami
URL	uniform resource locator, jednotná adresa zdroje
FQDN	fully qualified domain name, plně specifikované doménové jméno počítače
CSV	comma separated values, hodnoty oddělené oddělovačem
XML	extensible markup language, rozšiřitelný značkovací jazyk
JSON	JavaScript Object Notation, formát zápisu dat

1 Úvod

Elektrická energie se stala součástí života běžného člověka. Na jedné straně stojí spotřebitel, jehož přáním je, aby si mohl do elektrické zásuvky zapojit svůj spotřebič, a to kdykoli a na libovolně dlouho. Na straně druhé stojí distribuční společnost, a potažmo vše s tím související od výroby až po přenos elektrické energie, jejímž úkolem je zajištění přepravy vyrobené elektrické energie ke konečnému spotřebiteli. Zde vstupuje v úvahu nespočet dalších postupů, principů, výpočtů aj. za účelem bezchybného zajištění přenosu elektrické energie. Jedná se o multioborovou vědní problematiku se zastoupením matematiky, fyziky, informatiky, energetiky aj.

Právě zmíněná multiobornost se do značné míry projevuje i v této diplomové práci. Jejím smyslem je bližší seznámení s problematikou výpočtu ustáleného chodu ES pro následné potřeby návrhu platformy, způsoby jak výpočty prakticky realizovat, a vhodným návrhem platformy, na které tyto operace budou prováděny. Svoje zastoupení tak zde nachází matematika i informatika, která je zde však převažujícím prvkem zájmu. Proto je možné tvrdit, že smyslem této práce je vytvořit takovou softwarovou platformu, na které bude možné provádět vybrané typy úloh nad schematem ES. Pro dokreslení kontextu pak bude prakticky implementována výpočetní úloha řešící ustálený chod ES.

Jak je patrné ze samotného názvu této práce, tak nemalá část je zde věnována současným technologiím v prostředí webu. Důvodů je hned několik. Jedním ze zásadních je fakt, že současné programy řešící stejnou problematiku jsou postavené pro desktopový provoz a z velké části postrádají multiplatformní řešení. Nelze tvrdit, že webové technologie jsou absolutně multiplatformní. V poslední době však jejich vývoj k tomuto směřuje. Proto je vhodné prostřednictvím této práce prozkoumat, nakolik jejich vývoj pokročil a zda-li jsou použitelné. S webovými technologiemi souvisejí i následující otázky. Jak se změní architektura systému zavedením webových služeb? Je možné nahradit desktopovou aplikaci aplikací webovou? Pokud ano, pak v jakém rozsahu? Je možné uvažovat nasazení takto upraveného systému (ve smyslu použití webových technologií) do produkčního prostředí?

Samotná práce je systematicky rozdělena do několika částí. V úvodu je stručně popsán stávající stav. Na jeho základě je poté proveden návrh platformy, aby splňoval praktické požadavky vznešené od konzultanta a aby naplňoval zadání této práce. Následující kapitoly se postupně zabývají bližším popisem jednotlivých komponent platformy.

2 Analýza stávajícího stavu

Diplomová práce navazuje na předcházející magisterský projekt (viz [1]), který se zabýval otázkou, jak uchovávat v počítači reprezentaci ES. Jelikož se jedná o téma vysoce odborné, celá práce byla konzultována s pracovníky oddělení Provozu řídících systémů Východ – Hradec Králové ze společnosti ČEZ Distribuce a.s.

Jak již bylo naznačeno v úvodu, tato práce se zabývá oblastí plánování a řízení sítí. Jelikož se jedná obecně o dosti široké téma, práce se v teoretické rovině zaměřuje pouze na výpočet ustáleného chodu ES. A právě touto problematikou, a mimo jiné i řadou dalších, se zabývají další komerčně dostupné systémy. Jedním z nich je i systém RIS – SCADA/EMS od společnosti ELEKTROSYSTEM, a.s. (viz [2]). Z rize praktického důvodu, kterým je přítomnost tohoto systému v ostrém provozu u konzultanta, bude i následně uvažováno použití právě tohoto systému.

2.1 Popis elektrizační soustavy

Hlavní úlohou ES je přenos elektrické energie z místa výroby (výrobci) do místa spotřeby (spotřebitelé). Je tvořena souhrnem vedení, které může vést vzduchem, pomocí různých typů stožárů nebo se může jednat o kabelovou variantu. Dále je dělena na přenosovou síť, která vytváří přenosovou cestu mezi centry výroby a spotřeby, a distribuční síť, která slouží k přenosu elektrické energie ke koncovým spotřebitelům.

ES je v počítači reprezentována schematem. **Schema** je množina prvků, které dohromady vytvářejí ES. Samotné prvky soustavy lze rozdělit do tří kategorií – **úseky**, **uzlové body** a **objekty**. Pro řešení úlohy ustáleného chodu ES jsou použity pouze první dvě kategorie prvků.

Úsek je takový prvek soustavy, který spojuje alespoň dva rozdílné uzlové body a může nabývat typu:

- vedení,
- transformátor,
- trojvinuťový transformátor,
- reaktor,
- spínací prvek.

Uzlový bod je takový prvek soustavy, který může být propojen s ostatními uzlovými body za pomoci úseku a může nabývat typu:

- napájecí bod,
- odběrný bod,
- turbogenerátor,
- hydrogenerátor,
- nadřazená soustava.

Objekt je takový prvek soustavy, který seskupuje vybrané uzlové body. Pro účely řešení úloh neposkytuje žádné relevantní informace. Jeho použití je zejména pro organizaci uzlových bodů.

Jedním z výstupů zmíněného magisterského projektu (viz [1]) byl jazyk popisující schema sítě a jeho prvky. Byl vytvořen v jazyce XML, který i přes svojí košatější strukturu velmi přesně vyjadřuje vztah jednotlivých prvků a jejich atributů. Právě z takto popsaného schematu soustavy bude dále možné vycházet při návrhu zmíněné platformy.

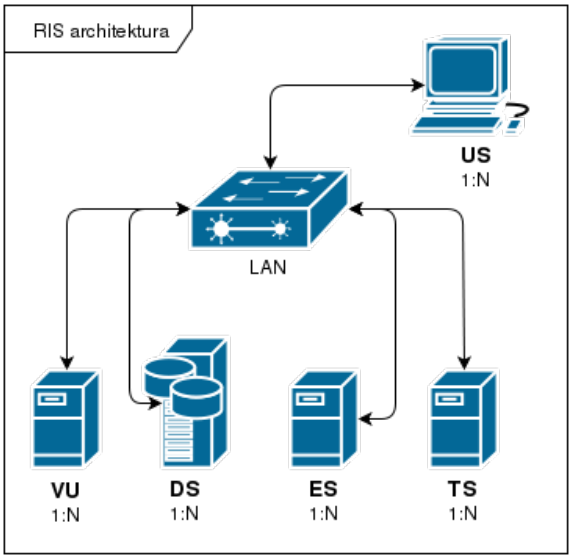
2.2 Systém RIS a jeho architektura

Řídící systém RIS – SCADA/EMS vytvořila společnost ELEKTROSYSTEM, a.s. (viz [2]). Jedná se o komerční produkt, který je navržen především pro distribuční společnosti s potřebou řízení jejich distribuční sítě. Obsahuje mnoho funkcionalit a komponent, mezi kterými je např. editor schematu soustavy, který umožňuje vytvářet i editovat soustavy. Další jeho komponenty pak řeší estimaci chodu sítě, kontingenční analýzu, chod sítě, výpočty zkratů, výpočty topologie, ale i samotné řízení sběru dat z energetických objektů. Nabízí také funkcionalitu dispečerského řízení zvolené sítě. Systém se skládá z několika hlavních částí, které jsou zachyceny na obrázku 2.1. Každá jednotlivá část musí být ve funkčním systému zastoupena nejméně jednou instancí, což je na uvedeném obrázku vyjádřeno symbolicky jako *1:N*. Patří mezi ně:

- datové servery (DS), které zajišťují chod systému,
- servery vzdálených uživatelů (VU), které zajišťují vzdálené připojení do podnikové sítě,
- telemetrické servery (TS), jejichž hlavní náplní je sběr dat ze vzdálených objektů,
- servery vyšších energetických funkcí (ES), které realizují numerické výpočty vyšších funkcí,
- uživatelské stanice (US), pomocí kterých lze systém obsluhovat.

Z obrázku 2.1 je dále patrné, že celý tento systém používá pro komunikaci TCP/IP protokol. Vzhledem k potřebě spojení objektů, které jsou různě geograficky umístěné, se pro účely spojení používají různé technologie od dedikovaných okruhů počínaje až po VPN připojení. To zajistí, že veškeré objekty, které podléhají vzdálenému řízení, vytváří v distribuční oblasti (Východní Čechy) privátní, tzv. podnikovou, síť. Ta je pak dále spojena s ostatními distribučními sítěmi v rámci distributora.

Z dokumentace též vyplývá, že systém RIS není multiplatformní. Převážná většina serverů je provozována na operačním systému Linux. Existují dílčí programy, které jsou spustitelné pouze pod operačním systémem Windows. Vzhledem k tomu, že se jedná právě o stěžejní jádro celého systému, bez kterého se žádná z uživatelských stanicic neobejde, je namístě konstatování oné nemultiplatformosti. Výjimkou jsou pak uživatelské stanice ve formě desktopových aplikací, které by měly fungovat, jak na operačním systému Linux, tak i Windows.



Obrázek 2.1: Architektura systému RIS

3 Návrh platformy

Platforma je definována jako prostředí po stránce hardwarové i softwarové, které umožňuje bezproblémový chod aplikací na ní provozovaných. Není bez zajímavosti, že v dnešní praxi se zejména pro komplexnější aplikace či úlohy toto slovo vyskytuje poměrně často. Mnohdy je totiž potřeba sladit činnost více aplikací, které by měly právě poskytovat určité běhové prostředí. A tím jsme se dostali zpátky ke slovu platforma. Je možná ambiciózní toto slovo používat. Zvláště v kontextu diplomové práce a za podmínek ve kterých vznikala. Nicméně právě toto slovo nejlépe vystihuje přesně tu množinu aplikací, funkčnosti, nápadů a řešení, kterými se diplomová práce dále zabývá.

3.1 Požadavky na funkčnost

Mezi základní požadavky na funkcionalitu platformy, které samozřejmě vycházejí z reálné potřeby a konzultací, jsou zařazeny následující.

1. Navrhnout a vytvořit jako volně šiřitelný software a využít volně dostupné technologie (i z hlediska případného budoucího komerčního využití).
2. V případě vhodnosti použít webové technologie.
3. Všechny aplikace v rámci platformy vytvořit již od počátku s distribuovanou architekturou tak, aby byly dobře škálovatelné. Klást důraz na multiplatformní řešení.
4. Identity uživatelů spravovat centrálně spolu s jejich přístupovými údaji včetně možnosti vytváření účelových přístupových údajů ke konkrétním aplikacím.
5. Provádět autentizaci a autorizaci prostřednictvím centrálního přístupového bodu. Zde klást důraz na ochranu přístupových údajů a jejich sdělování pouze důvěryhodné autentizační straně.
6. Vyžadovat přístupové oprávnění k přístupu do schematu sítě a k zadávání výpočetní úlohy vytvořené nad zvoleným schematem. Žádné další omezení přístupu k jednotlivým aplikacím či jejich částem není požadováno.
7. Vytvořit editor schematu sítě s možností editace prvků, grafické vizualizace schematu a zadávání výpočetních úloh. Jiné požadavky na jeho UI nejsou.

Nicméně je nutné při jeho návrhu brát v potaz přehlednost a srozumitelnost pro koncového uživatele.

8. V případě použití webových technologií jako UI vždy klást důraz na responzivní design.

9. Provést import dat ze systému RIS. Není jej třeba integrovat do editoru, postačí spouštění z terminálu a následný import provedený z předávacího formátu. Uvažovat možné budoucí rozšíření v implementaci jiného formátu.

10. Implementovat výpočetní úlohu ustáleného chodu sítě.

3.2 Potenciální uživatel

Potenciálních uživatelů je několik skupin. Společným zájmem následně identifikovaných skupin uživatelů je skutečnost, že tato platforma je koncipována jako volně šiřitelná s otevřeným zdrojovým kódem. Přestože je pravdou, že u takto založených projektů rozhoduje o jejich přežití podpora ze strany komunity či alespoň vůle vývojářů pokračovat ve vývoji, tak i přes tuto skutečnost se jedná o ve své oblasti nezanedbatelný krok kupředu a je jen na každém uživateli, zda si projekt osvojí.

První skupinou jsou **distribuční společnosti** provozující distribuční soustavy. Vzhledem k faktu, že tato práce vznikala za pomoci konzultací s pracovníky právě takové společnosti, je zřejmé, že toto je jeden z potenciálních uživatelů. Byť se nejedná v žádném případě o náhradu již používaných dispečerských a jiných systémů, tak tato platforma si může najít své uplatnění jako alternativní nástroj pro náročnější matematické výpočty vybraných typů úloh, které si její provozovatel může sám modifikovat dle svých přání. Ne každému totiž může vyhovovat uzavřenost dnes používaných systémů.

Druhou skupinou jsou pak samostatní **projektanti** elektrických sítí či společnosti zabývající se tímto oborem. I v tomto případě hraje roli potřeba provádění výpočtů vybraných typů úloh. Ve spojení s různými možnostmi provozu platformy, možnostem k jejím úpravám (samozřejmě v případě potřeby) a své dostupnosti, se může jevit jako zajímavá možnost.

Následující skupina se od předchozích odlišuje tím, že se ve své podstatě nejedná o uživatele platformy, nýbrž o jejího **provozovatele**. Tuto platformu by mohl provozovatel provozovat i jako SaaS, kdy zákazník využívá služby platformy a její provozovatel si na ní tímto způsobem realizuje podnikatelský záměr. Pro účely této práce však tato možnost rozhodně není prioritní a jedná se tak pouze o úvahu.

Poslední skupinu tvoří lidé či instituce, které tuto platformu využijí k **edukativním účelům**. Ne že by nebylo možné ji provozovat z edukativních účelů i pro výše uvedené typy uživatelů, ale u nich se předpokládá spíše praktické využití na základě reálných potřeb.

3.3 Model platformy

3.3.1 Komponenty

Na základě požadavků uvedených v kapitole 3.1 jsou definovány základní komponenty platformy. Jak vyplývá z třetího požadavku, všechny komponenty jsou navrženy s ohledem na distribuovatelnost. Tím je myšleno, aby každá z komponent byla připravená na škálovatelnost, na zvýšení jejího výkonu, kapacity, či prostého fyzického oddělení od zbytku platformy v případě potřeby.

První takovou komponentou je **Poskytovatel identit** (**IdP**). Jejím úkolem je správa identity jednotlivých uživatelů a k ní příslušející přístupové údaje. Další neméně významnou funkcí je provádění ověřování uživatelů, kteří se pokoušejí k platformě přistoupit a udělit jim patřičná oprávnění. A v okamžiku příchozího požadavku vždy ověřit, že daný požadavek a jeho původce, uživatel, smí daný požadavek provést.

Další komponentou, z požadavků vyplývající, je **Editor**. Prostřednictvím komponenty editoru získá uživatel přístup k uloženým datům, a to ať už v textové či grafické podobě. Textovou podobou je myšlena možnost náhledu a úprav pomocí formulářů či jiných k tomu potřebných prostředků tak, aby měl uživatel možnost s daty pracovat. Grafickou podobou je myšlena možnost vizualizace vybraných dat. V tomto případě lze uvažovat vizualizaci vytvořeného schematu sítě, jelikož to obsahuje všechny použité prvky soustavy a dává ucelený pohled na ní samotnou prostřednictvím grafu sítě.

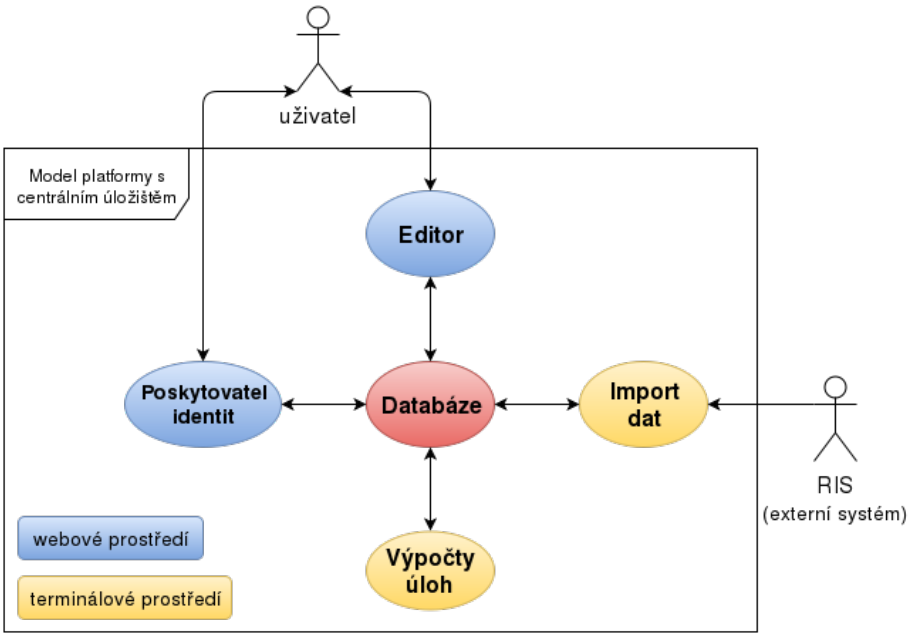
Spolupráci s externími programy, prozatím ve formě přebírání dat z externího systému, řeší komponenta **Import dat**. Tato funkcionalita je dostupná z toho důvodu, aby bylo možné z externího systému importovat již existující data. V tomto případě data o schematu sítě. Vzhledem k faktu, že je zde kladen důraz na jistou modularitu, tak tato komponenta je navržena způsobem, aby se dala upravovat pro různé konkrétní programy. Praktická implementace je zaměřena na systém RIS.

Poslední komponentou vyplývající z požadavků jsou **Výpočty úloh**. Vzhledem k faktu, že existuje více typů výpočetních úloh, které lze nad daným schematem sítě provádět, a s přihlédnutím k výkonovým možnostem různých dále popsaných variant, je zapotřebí klást u této komponenty důraz na modulárnost, škálovatelnost a vhodné techniky jejího zpracování.

3.3.2 Úložiště dat

Požadavky v kapitole 3.1 nikterak nespecifikují způsob uložení dat a jejich případné sdílení napříč zmíněnými komponentami. Toho lze dosáhnout různě.

První variantou je, že každá komponenta má **vlastní úložiště dat**, které spravuje. Tento přístup má relativní výhodu v tom, že nepřidává do platformy další komponenty. Nicméně větší nevýhodou se jeví jejich vzájemné provázání a následná správa. Druhá varianta tento problém řeší za pomoci **centrálního úložiště dat** pro veškeré komponenty. To je realizováno prostřednictvím databáze. Tento případ je zachycen na obrázku 3.1.

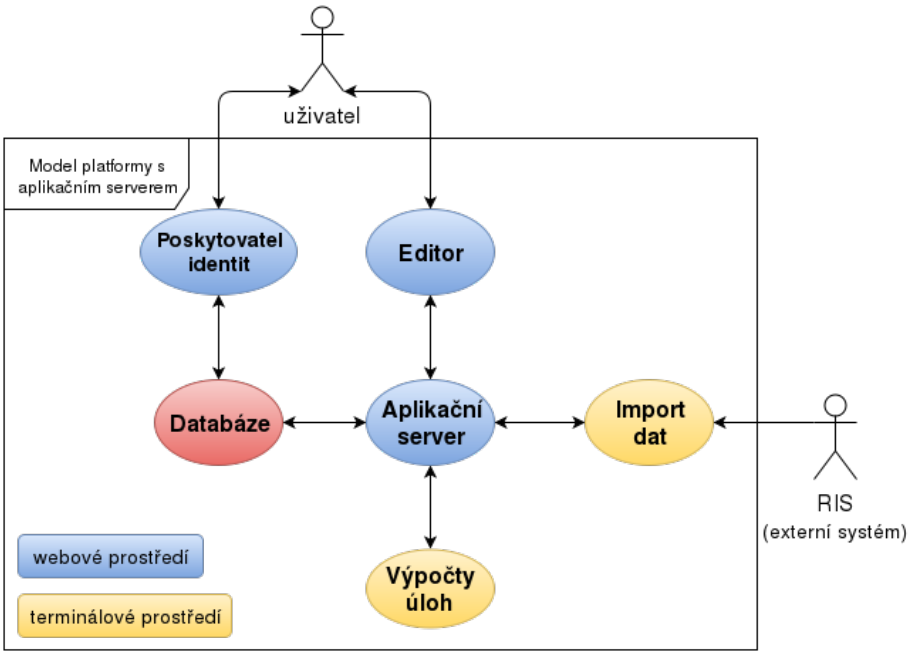


Obrázek 3.1: Model platformy s centrálním úložištěm

Avšak i pro tento případ existuje negativní důvod, proč jej nezvolit. Tím důvodem je závislost na konkrétní použité databázi. V případě její změny či výměny za jinou by bylo potřeba zasahovat do jednotlivých komponent. Přestože dnes není prakticky problém vytvářet konkrétní komponenty s dostatečnou abstrakcí a poté jen konfigurací nastavit konkrétní databázi, minimálně v tomto lze spatřovat onu nutnou modifikaci. Její výhodou se jeví rychlost zpracování požadavků, protože požadavky zpracovává přímo databázový stroj.

Třetí variantou je použití **aplikačního serveru** (**AS**). Tento případ je znázorněn na obrázku 3.2. Zde je patrné, že prostředníkem pro výměnu dat mezi komponentami (aplikacemi) a databází je aplikační server. Ten může podporovat různé způsoby zpracování požadavků a odpovědí a také jejich různé formáty. O této problematice blíže pojednává kapitola 5. V kontextu předchozího odstavce je nutné zmínit, že tato varianta může být v určitých situacích výkonově horší, protože místo přímého přístupu komponenty (aplikace) do databáze se ona komponenta obrací de facto na prostředníka, který ji požadovanou akci zprostředkuje. V dnešní době existují techniky, např. vhodné cachování dat, s nimiž lze tyto problémy minimalizovat. Na druhou stranu zde existuje minimálně jedna nepřehlédnutelná výhoda. Tou je možnost specifikace daného rozhraní nezávisle na použité databázi a odstínění konkrétní komponenty od nutnosti přesně implementovat danou databázi. V praxi se jedná o to, že aplikační server poskytuje rozhraní ve formě definovaných struktur volání či přístupu k datům. Tato volání pak provádí komponenta, která nemusí řešit, jak se data interně ukládají. Pro ni je pouze důležité, aby dokázala pracovat s daným rozhraním, které ji zpřístupňuje aplikační server. To přináší i druhou výhodu. Je předem jasné, čím aplikační server disponuje a co naopak nepodporuje.

Ke každému takovému rozhraní by se měla vytvářet dokumentace, aby programátor věděl, co a jak může v rámci služeb aplikačního serveru využít.



Obrázek 3.2: Model platformy s aplikačním serverem

Na základě předchozího srovnání modelu bez aplikačního serveru a s jeho použitím bylo zvoleno řešení, které jej využívá. Krom výše uvedených výhod tohoto řešení lze přidat ještě jednu další. Tou je skutečnost, že aplikační server lze vytvořit i jako webovou službu, resp. aplikační server poskytující webové služby.

3.3.3 Realizovatelnost komponent

Vzhledem k předešlé volbě aplikačního serveru je možné konfrontovat tuto skutečnost s výše uvedenými komponentami. Smyslem této konfrontace je pouze zjistit, zda-li je daná komponenta realizovatelná vůči aplikačnímu serveru.

První zmíněnou komponentou je **Poskytovatel identit**. Z obrázku 3.2 je patrné, že ten přímo s aplikačním serverem nekomunikuje. Přesto je zde jeden jediný okamžik, kdy dochází k nepřímé komunikaci. Tímto je okamžik příchozího požadavku na aplikační server, kdy poskytovatel identit ověří, zda je požadavek validní po stránce autorizační. Tohoto ověření se však může zhostit aplikační server sám, jelikož sám má přístup k databázi uchovávající autorizační záznamy. Autorizační záznam musí vždy vytvořit poskytovatel identit. Následné ověření spočívá pouze v kontrole jeho existence, což je možné provádět aplikačním serverem. Je možné konstatovat, že tato komponenta je realizovatelná. Lze ji také vytvořit jako webovou aplikaci. Pro správu identit či přístup k platformě tato skutečnost nepředstavuje omezení.

Vytvoření **Editoru**, jako webové aplikace, je bez jakéhokoli omezení, ba naopak navazuje v duchu webu a jeho služeb na předešlé. Je možné jej realizovat buď jako

klasickou webovou aplikaci nebo též jako tzv. single page aplikaci. Ta se vyznačuje tím, že aplikační server slouží pouze jako zdroj obsahu a veškerý kód aplikace samé je interpretován na klientovi v jeho prohlížeči. Co se výše uvedené funkcionality týká, toto řešení dokáže pokrýt veškeré požadavky. Tedy jak možnost práce s daty ve formě jejich úprav prostřednictvím formulářů, tak i vizualizaci schematu sítí.

Komponentu **Import dat** je třeba hodnotit ze dvou pohledů. Jednak je to implementace, která realizuje vlastní provedení importu dat nezávisle na GUI. Tato část se jeví jako nekonfliktní vůči aplikačnímu serveru a poskytovateli identit, protože pouze využívá jimi poskytované služby. Druhý pohled je pak samotné GUI. V požadavcích kapitoly 3.1 se pro tuto komponentu uvádí, že není potřeba se zabývat GUI, nýbrž že stačí pouze UI ve formě terminálového ovládání spouštěné aplikace. I tento pohled je nekonfliktní. Z toho tedy vyplývá, že komponenta import může být vytvořena jako terminálová aplikace ovladatelná příkazy terminálu. To je vlastně předstupněm pro desktopovou grafickou nadstavbu. Nyní se nabízí otázka, proč by to nemohla být webová aplikace. Samozřejmě že by jí mohla být. Úlohou této komponenty je importovat i rozsáhlá data. A to je v rámci webové aplikace přinejmenším značně diskutabilní. Taková aplikace by totiž potřebovala prakticky neomezený čas běhu jí samé. Tím je mířeno na interpretované jazyky na webovém serveru s nastavenou maximální dobou trvání daného skriptu. Právě ta by mohla způsobit jeho předčasné ukončení dříve než se dané operace dokončí. Je tu možnost toto nastavení upravit, nicméně je otázkou, zda-li by se i po této úpravě jednalo o vhodnou volbu třeba právě z hlediska výkonu.

Poslední komponentou je komponenta **Výpočtů úloh**. Jejím úkolem je, jak již název napovídá, provádění výpočtů matematických úloh. Je zřejmé, že v první řadě hraje roli efektivita výpočtu spolu s dostupným výpočetním výkonem. Požadavky typu paralelizovatelnost či více vláknové zpracování jsou na předních příčkách. Právě z tohoto důvodu je prakticky vyloučené, aby výpočty prováděl webový prohlížeč uživatele. Přestože již dnes některé webové prohlížeče pracují tak, že každý otevřený panel běží ve svém vlastním procesu, stále se nedá mluvit o dostatečně dostupném výkonu. Těžko si představit uživatele, který si ve svém webovém prohlížeči, byť pouze v jednom panelu, spustí nějaký rozsáhlejší výpočet. Nemluvě o situaci, kdy je potřeba provádět více výpočtů současně. Proto lze konstatovat, že webové prohlížeče, resp. webové aplikace, jsou pro tento účel jen stěží použitelné a pro praktický běh platformy spíše nevhodné. V tomto směru má větší perspektivu tuto komponentu navrhnout jako systémovou službu, která se spouští na pozadí. Škálovatelnost je v tomto případě možná jak výběrem vhodného hardwaru, tak i prostým přidáváním výpočetních uzlů či přesunutí výpočtů do specializovaných clusterů. Webová aplikace by mohla dané výpočty ovládat, například prostřednictvím aplikačního serveru, ale nikoli provádět.

Tato kapitola se tak vlastně stala zdrojem odpovědí otázky položené v úvodu této diplomové práce. Odpovídá na stěžejní otázky, kdy, kde a zda má smysl použít webové technologie. Jejich použití je popsáno v kapitolách jednotlivých komponent.

3.4 Datový model

V návaznosti na předchozí kapitolu, která dospěla k závěru, že vhodnější je zvolit model s aplikačním serverem, je nyní zapotřebí alespoň v základním rámci definovat strukturu úložiště dat. To vše samozřejmě s přihlédnutím k již získaným poznatkům o struktuře jednotlivých prvků schematu sítě a schematu samotnému, které byly zjištěny a popsány v magisterském projektu (viz [1]).

Navrhnout vhodný model lze více způsoby. Avšak cílem práce není se zabývat zdaleka všemi možnostmi. Proto je zde věnován prostor pro dva, byť na první pohled odlišné, ale při bližším pohledu do značné míry shodné, modely.

3.4.1 Varianty datového modelu

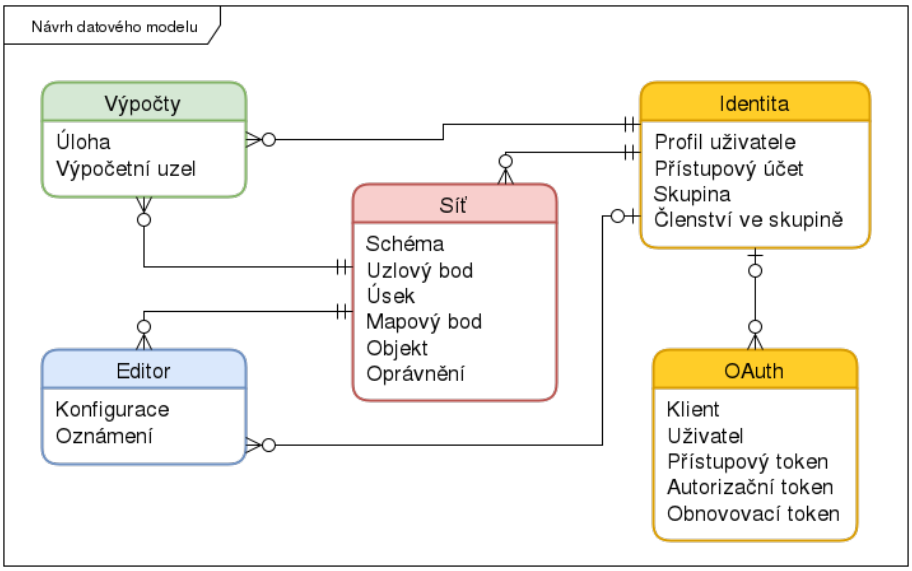
První přístup k vytvoření modelu spočívá ve vytvoření jedné jediné databáze, ve které by byly reprezentovány všechny potřebné entity s patřičnými vzájemnými relacemi. To tedy znamená, že databáze je jeden logický celek, který interně obsahuje jednotlivé entity. Z pohledu výkonnostního je zřejmé, že databázový stroj je nutné provozovat v dostatečně výkonném prostředí. Toho lze dosáhnout jednak hardwarově na samostatném serveru, ale i za použití databázového clusteru, kdy existují různé techniky, jak rozložit zátěž kladenou na cluster. Pro ilustraci lze uvést architekturu *Master-Slave*, kdy uzel typu *Master* provádí úpravy uložených dat a distribuuje tyto úpravy dále na uzly typu *Slave*, které slouží pro jejich získávání. Dalším typem je pak cluster s architekturou *Multi-Master*, který obsahuje uzly typu *Master* a data jsou vzájemně mezi všemi replikována.

Druhým přístupem je seskupení entit do logických celků, které jsou reprezentovány samostatnými databázemi. Tento přístup skýtá možnosti větší distribuovatelnosti než předchozí, jelikož lze, ať už v rámci jednotlivých serverů, či přímo clusterů, fyzicky oddělit jednotlivé databáze. Nicméně právě tento přístup k návrhu je náročnější, což vyplývá z onoho oddělení jednotlivých databází. V případě provozu takovýchto databází v rámci jednoho databázového serveru či databázového clusteru je toto řešení praktické. Není však příliš vhodné pro situaci, kdy budou jednotlivé databáze provozovány separátně a bez propojení s ostatními. To z toho důvodu, že by pak byly omezeny dotazy nad vícero entitami z různých databází.

3.4.2 Návrh datového modelu

Následující obrázek 3.3 zachycuje model vytvořený druhým přístupem. Je z něj patrná existence několika databází, kdy každá z nich obsahuje logicky spřízněné entity. Je však nutné na tomto místě zmínit, že tento obrázek pouze schematicky zobrazuje vztah jednotlivých databází a nastiňuje entity, které jsou v nich umístěny. Nejedná se tedy o ER diagram v pravém slova smyslu, nýbrž pouze o vyjádření logických vazeb mezi databázemi za pomoci symboliky z ER diagramu.

Databáze **Síť** obsahuje entity popisující schema sítě. Jedná se tedy např. o schema, úseky, uzlové body, objekty, atd. Další databází je pak **Identita**. Ta má uchovávat data související s identitou uživatele a jeho přístupovým účtem. V těsné spolupráci



Obrázek 3.3: Návrh datového modelu

s databází Identita funguje databáze **OAuth**, která obsahuje entity související s autorizací uživatelů. Samotný princip autorizace je následně uveden v kapitole 4.3. Pro potřeby správy jednotlivých výpočetních úloh slouží databáze **Výpočty**. Poslední databází je **Editor**, který uchovává data a nastavení editoru.

Detailní struktura jednotlivých databází je zachycena v přílohách. ER diagram v příloze A zobrazuje databázi poskytovatele identit, v příloze B databázi OAuth, v příloze C databázi sítě a v příloze D databázi výpočtů.

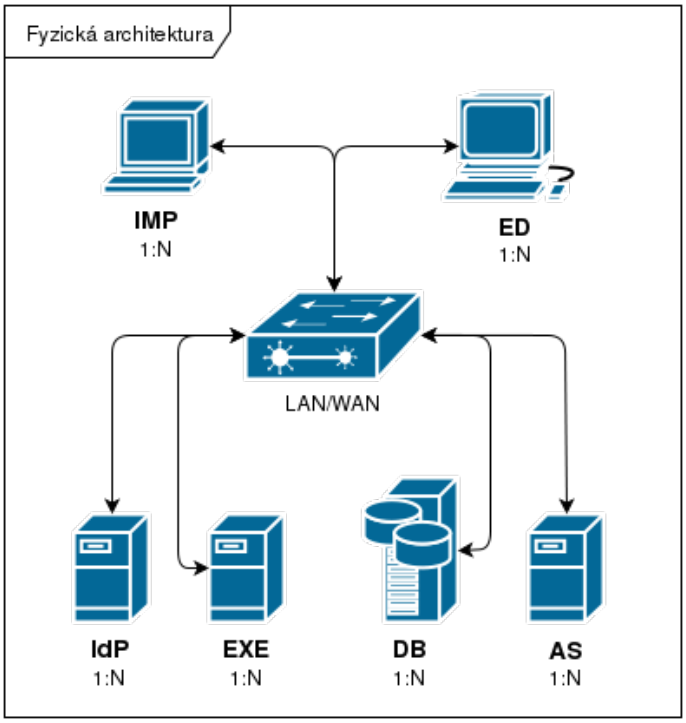
3.5 Provozní prostředí

Z předchozích kapitol je již zřejmé, jaké komponenty by měla vytvářená platforma obsahovat. Nyní přichází na řadu grafické vyjádření architektury této platformy tak, aby bylo zřejmé, jaké fyzické celky obsahuje a jak by měly být uspořádány.

3.5.1 Fyzická architektura platformy

Na obrázku 3.4 je zachycena fyzická architektura platformy. Je zde patrné, že veškeré její komponenty jsou připojeny do místní (LAN) či veřejné (WAN) sítě. Tím je naznačeno, že tato platforma nutně nemusí být umístěna v jedné lokalitě, nýbrž lze jednotlivé komponenty dislokovat i do jiných lokalit. To je umožněno jednak způsobem komunikace, kde všechny komponenty spolu komunikují prostřednictvím TCP/IP protokolu a za druhé skutečností, že je takto platforma navržená. Komponentami typu server zde zobrazenými jsou: aplikační server (AS), databázový server (DB), výpočetní server (EXE), poskytovatel identit (IdP). Komponenta editoru (ED) je pro tyto účely zařazena spíše do kategorie klientských aplikací, přestože je jako webová aplikace hostována na webovém serveru. A poslední komponentou,

pro tento okamžik rovněž typu klientské aplikace, je import (IMP). Pod každou z komponent je uvedeno její minimální vs maximální zastoupení v rámci platformy. Na obrázku mají komponenty vyznačeno zastoupení formátu 1:N, což znamená, že každá z komponent tam bude minimálně jednou. Pro demonstrační účely a ověření funkčnosti je takovéto zastoupení vhodné. Pro praktické využití by bylo možné nalézt i jiné hodnoty. Pokud by kupříkladu bylo dopředu známo, že platforma nepotřebuje importovat data z externích systémů, pak by logicky tato komponenta nebyla zastoupena. Na druhé straně je ale zjevné, že se zde vyskytují i komponenty, u nichž takováto diskuze postrádá smysl, jelikož bez nich by platforma nebyla funkční. V tomto smyslu by se mohlo jednat o aplikační a databázový server, případně pak o poskytovatele identit a výpočetní server.



Obrázek 3.4: Fyzická architektura platformy

3.5.2 Platforma jako SaaS

Pro účely provozního prostředí je ještě vhodné vzpomenout pasáž, která v kapitole 3.2 pojednává o možnosti provozovat poskytovateli platformu modelem SaaS.

SaaS (software as a service či česky software jako služba) je model nasazení softwaru, kdy dochází k hostování aplikace provozovatelem služby. Služba je dále nabízena uživatelům přes Internet. Uživatel je tak oproštěn od kupování softwaru či hardwaru, instalaci, konfiguraci a provozu technologií jako takových. SaaS vznikla jako reakce na potřebu snižování nákladů na software, rychlého nasazení a outsourcingu. Nasazením SaaS odpadají investice do aplikačního softwarového balíku.

Zákazník jednoduše využívá softwarovou funkcionalitu na dálku jako službu. Zákazník platí jenom provozní náklady. [3]

U tohoto modelu je možné namítnout, že tím uživatel může ztratit kontrolu nad svými daty, protože je de facto svěřuje třetí straně. Zde záleží, jakým způsobem provozovatel dále chrání data uživatelů. V dnešní době, kdy si i velké nadnárodní technologické firmy jsou tohoto problému vědomy a kdy existuje velké množství způsobů zabezpečení dat, se ukazuje, že to není problém technický. V tomto kontextu je nadále zabezpečení dat touto prací navrhovanou platformou uvažováno a je popsáno u jednotlivých komponent v následujících kapitolách.

3.6 Metodiky vývoje softwaru

Vývoj softwaru prochází od svého vzniku změnami. Smyslem metodik pro jeho vývoj je stanovení postupů a způsobů, jak takový software vhodně vyvíjet. Avšak i zde platí, že sebelepší metodika nemusí vyhovovat všem případům či vývojářům. Proto je smyslem této kapitoly poukázat na některé významné existující metodiky.

Následující metodiky se řadí do kategorie tzv. **modelu životního cyklu**. Tedy pomocí modelu je definována posloupnost jednotlivých etap vývoje bez stanovení metody, pomocí které se boudou etapy realizovat.

Model programuj a opravuj je nejprimitivnější model vývoje softwaru. Jedná se o jednoduchý způsob vývoje, jež je založen na opakujících se činnostech programování, testování a opravě chyb. V první etapě jsou specifikovány požadavky, následně se přistupuje k opakujícímu se cyklu kódování a opravování.

Vodopádový model je sekvenční vývojový proces, ve kterém lze přirovnat vývoj k neustále se svažujícímu toku. Základními fázemi jsou: analýza požadavků, návrh, implementace, testování, a integrace. Projekt se organizuje do fází, které se řadí postupně za sebou. Často bývá v praxi doprovázen až formální administrativou, která dokumentuje změny požadavků, jejich schválení, atp. Nevýhodou může být, že testování probíhá po ukončení vývoje a tudíž má v principu pouze potvrdit správnost vyvinutého softwaru. [6]

Spirálový model byl definován Barry Bohem v roce 1985. Tento model životního cyklu klade především velký důraz na analýzu rizik. Model rozděluje projekt do jednotlivých kroků. V každé iteraci se provádějí následující fáze: stanovení cílů, analýza rizik, návrh řešení, vývoj, testování a plánování. Zde, na rozdíl od vodopádového modelu, jsou výstupy po dokončení každé iterace předávány ve formě prototypu, kterým si zákazník může snáze ověřit, že se vývoj ubírá správným směrem. Výhodou je snížení nákladů na vývoj softwaru tím, že jsou včas odhalovány chyby. [6]

Další kategorií jsou pak metodiky založené na **objektovém přístupu**, které nahlížejí na data a funkce jako na součást objektu. Tím se snaží prostřednictvím předepsaných modelů důvěryhodněji zachytit modelovanou realitu.

Rational Unified Process (RUP) představuje proces, který využívá přístup k vývoji softwaru za pomoci řad nástrojů, šablon, artefaktů a včasných dodávek prototypů. Od ostatních výše zmíněných přístupů se převážně liší tím, že se jedná o rozsáhlou a detailněji propracovanou metodiku. Metodika RUP je vhodná primárně

na velké projekty, nicméně je možné ji upravovat i pro jednodušší projekty. Vývoj probíhá v iterativních cyklech, kde každá iterace má podobu menšího vodopádu, tj. obsahuje všechny fáze tohoto modelu (analýza – testování). Na počátku projektu je nejvíce času věnováno specifikaci požadavků. Pozdější iterace vývoje se zaměřují na implementaci a testování. Výsledkem každého cyklu je nová verze produktu. [5]

Neméně zajímavou skupinu tvoří **agilní metodiky**. To jsou metodiky založené na vývoji softwaru na iterativním a inkrementálním přístupu. Umožňují rychlý vývoj softwaru a zároveň dokáží reagovat na změnu požadavků v průběhu vývojového cyklu. Podle těchto metodik se správnost systému ověří jedině pomocí rychlého vývoje, předložení zákazníkovi a následných úprav dle zpětné vazby. Poměrně novým a známým zástupcem této kategorie je **Scrum** (viz [7]) či **Extrémní programování** (viz [8]).

Do jisté míry odlišným způsobem přístupu k vývoji se následující technika vývoje liší od předchozích. Předchozí způsoby vývoje se zaměřovaly na situaci, kdy byl kód programu napsán a bylo třeba jej otestovat. Avšak existuje i opačný přístup, kdy je nejprve napsán test pro určitou část programu a ta je až následně vytvořena. Tento přístup se nazývá **TDD** (Test-driven development či česky programování řízené testy). Jeho smyslem je v prvé řadě tvorba testů a až následně vývoj. Samotná tvorba testů je samozřejmě založena na dostupných technikách testování uvedených v následující kapitole.

Vzhledem k povaze této diplomové práce a skutečnosti, že na vývoji se v současné době podílí pouze jeden člověk, tedy autor, je nasnadě, že se vývoj blíží vodopádovému modelu.

3.7 Techniky testování softwaru

Stejně jako v předchozí kapitole existuje řada metodik, tak i zde existuje více pohledů a způsobů dělení, jak software testovat. Z předchozí kapitoly je patrné, že tematika testování byla součástí přinejmenším většiny uvedených metodik. I když u některých se mohlo jednat spíše o formální stránku věci, u některých byla použita v pravém slova smyslu a ulehčila vývoj.

Prvním pohledem na testování je pohled pomocí **způsobu provedení testu**. Nejjednodušší je metoda pomocí **černé a bílé skříňky**. Testování černé skříňky je založeno na myšlence, že tester není seznámen s vnitřní logikou testovaného softwaru. Poté tedy testování probíhá sledováním vstupů a výstupů, kdy se kontroluje, zda je na výstupu produkován očekávaný výsledek. Při testování bílé skříňky tester zná obsah zdrojových kódů a má možnost tak otestovat všechny situace, na které je daný software připraven. To může být v některých situacích kontraproduktivní, jelikož mu může uniknout nějaká podmínka, na kterou ani vývojář nepomyslel. V tom případě je výsledek takového testu nepřesný. Další možnost rozdělení technik testování je **manuální a automatizované testování**. Rozdíl spočívá v tom, že při manuálním testování testuje člověk, zatímco při automatizovaném testování za něho testuje software. Poslední v této kategorii je testování **splněním a selháním**. Testování splněním při začátku programu nejprve otestuje jeho elementární funkčnost. Pokud

program toto splní, pak tímto testem projde. Účelem není pokoušet se odhalit jeho slabá místa. Jedná se spíše o simulaci běžného použití systému. Naopak test selháním se provede až tehdy, když je zabezpečena elementární funkčnost. V této fázi již testujeme neobvyklé testové případy, stabilitu systému atd. [4]

Dalším pohledem na testování je rozdělení podle **úrovně vývoje**, čímž je myšleno v jaké úrovni se nachází samotný testovaný objekt. Prvním zástupcem této kategorie a též zástupce na nejnižší úrovni je **testování programátorem**. Programátor po vytvoření programového kódu svůj kód ihned ověří. V praxi si však programátor netestuje svoji část kódu, ale realizuje se tzv. *test čtyř očí*. Což je situace, kdy kód otestuje jiný programátor než ten, který jej vytvořil. Program je v tomto stupni kontrolován na úrovni zdrojového kódu. Dalším zástupcem je **jednotkové testování**. Tím je myšleno testování jednotlivých metod a funkcí daných objektů. Testovanou jednotkou je samostatně testovatelná část aplikačního programu. Testy těchto jednotek se zapisují ve formě programového kódu. V době, kdy je vývojář hotov se svými testy, přichází na řadu testovací tým. **Integrační testy** nepřipravuje programátor, ale především onen tým. Musí být ověřena bezchybná komunikace mezi jednotlivými komponentami uvnitř aplikace. Integraci lze ověřovat nejen mezi komponentami, ale také mezi komponentou a operačním systémem, hardwarem či rozhraním různých systémů. V této fázi se testuje integrace dosud jednotlivě ověřených částí. Po provedení integračních testů přichází na řadu **systémové testování**. Během těchto testů je aplikace ověřována jako funkční celek. Z toho tedy vyplývá, že se jedná o testování až v pozdější fázi projektu. Tyto testy ověřují aplikaci z pohledu zákazníka. Na připravených scénářích se simulují různé kroky, které v praxi mohou nastat. Obvykle probíhají v několika kolech. Nalezené chyby jsou opraveny a v dalších kolech jsou tyto opravy opět otestovány. Tyto testy jsou poslední fází, po které bude produkt předán zákazníkovi. Tato úroveň testů tak většinou slouží jako výstupní kontrola softwaru. Systémové testování je obsaženo prakticky v každém procesu testování. Bez této úrovně by celé testování softwaru nemělo žádný význam. V okamžiku, kdy zákazník převezme software, měl by jej podrobit **akceptačnímu testování**. To je prováděno podle připravených scénářů, které společně připravil zákazník s dodavatelem. Testy probíhají na testovacím prostředí u zákazníka. Nalezené nesrovnalosti mezi aplikací a specifikací jsou odeslány zpět vývojovému týmu, který provede jejich opravu. [9]

Způsob testování jednotlivých komponent v rámci platformy je u každé komponenty blíže popsán v příslušných kapitolách.

3.8 Licence a dostupnost

Veškeré kódy, skripty, programy aj. jsou vydány pod licencí MIT. Výhodou této licence je, že neomezuje použití takto licencovaného díla, pouze vyžaduje, aby byl text licence dále distribuován spolu s dílem, ať už jakkoli modifikovaným. Zdrojové kódy platformy jsou volně dostupné v GIT repozitářích (viz [10]). Repozitáře začínají prefixem **cpdn-***, kdy místo hvězdičky je uveden název jednotlivé komponenty.

4 Poskytovatel identity

V rámci kapitoly 3.1 byly jako jedny z prvních požadavků vzneseny ty směrem k identifikaci uživatele, jeho identitě a způsobu jejího ověření. K platformě smí přistupovat pouze ověřený uživatel. Proto je nejprve nutné provést jeho ověření. Tato problematika v sobě ukrývá tři základní okruhy – identitu uživatele, způsob jejího ověření (autentizace) a udělení oprávnění na základě identity (autorizace).

Bylo by možné použít již plně funkční systémy, které poskytují SSO, tedy tzv. Single Sign On, což lze přeložit jako jediné přihlášení na webu. Tím je myšleno, že uživatel se v rámci svého domovského poskytovatele identit autentizuje a ten mu následně vydá pověření, se kterým může pracovat v aplikaci třetí strany pod svojí identitou. Typickým zástupcem, a nejen v akademické obci, je projekt s názvem Shibboleth. Mohlo by se tedy zdát, že by bylo možné jej využít. To by beze sporu na určité části platformy bylo možné. Nicméně z hlediska toho, jak byla platforma navržena v kapitole 3.3, se zde nachází vhodnější varianta vlastní implementace autentizačního procesu a následné použití protokolu OAuth 2.0 jako autorizačního procesu. Toto spojení je vhodnější zejména z toho důvodu, že je přímo navrženo k zabezpečení a řízení přístupu aplikačního serveru jako poskytovatele webových služeb, což je přesně závěr výše citované kapitoly.

4.1 Identita uživatele

Identita uživatele je soubor informací, které popisují atributy uživatele jako osoby. Ty by v kontextu této platformy měly obsahovat základní údaje, jakými jsou jméno, příjmení, kontakt na osobu aj. Tyto údaje by měly v rámci celé platformy sloužit jako tzv. profil uživatele či profil uživatelovi identity. Profil je identifikován napříč všemi profily unikátním kódem, kterým je uživatelův email. Principiálně by to mohl být i např. nějaký náhodně generovaný řetězec nebo číslo. Z praktického hlediska však byl zvolen email. Je zcela na místě konstatování, že v rámci tohoto uspořádání by duplicitní záznamy nebyly vhodné.

V rámci identity daného uživatele by bylo možné ještě uvažovat rozšíření o jeho začlenění do skupiny uživatelů. To by bylo vhodné zejména pro situaci, kdy by s platformou pracovali uživatelé jednotlivých oddělení či společností. Byť toto rozšíření není stěžejním bodem této práce, je s ním alespoň uvažováno pro případ, aby takováto možnost mohla být v budoucnu implementována. Dalším rozšířením v tomto směru by mohla být správa jednotlivých skupin.

Naopak stěžejní částí spojenou s identitou uživatele, čili s jeho profilem, je

přístupový účet. Tento přístupový účet slouží pro ověření, že k platformě přistupuje právě ten uživatel, který zná svoji identitu a údaje pro její ověření, tedy přístupové údaje svého přístupového účtu. Tento účet nazvěme jako **hlavní přístupový účet**. Nad tímto účtem může být zřízeno monitorování, které dokáže ovlivnit politiku přístupu. Monitoring lze provádět u každého takového účtu, kdy lze uchovávat parametry jako čas přístupu, počet úspěšných/neúspěšných pokusů o přístup, z jaké IP adresy byl přístup proveden atd. Následně na základě těchto informací může být provedeno vyhodnocení, které může ovlivnit politiku přístupu.

V příloze A je přiložen ER diagram databáze pro uchovávání identit uživatelů a s nimi spojených přístupových účtů. Mezi hlavní entity patří entita **Profile**, která spolu s entitou **Contact** uchovává atributy uživatele a vytváří tak jeho identitu v rámci platformy. Druhou významnou entitou je **Account**, která spravuje hlavní přístupové účty. V této entitě je základním atributem email a heslo. Uživateli je na základě jejich znalosti umožněn přístup k dalším krokům ověření a udělení oprávnění k přístupu. Dále jsou zde uvedeny entity související se zmíněnou podporou organizace uživatelů do skupin či pro monitorování pokusů o přístup.

4.2 Autentizace

Autentizace je bezpečnostní opatření, které zajišťuje ochranu před falšováním identity, kdy se subjekt vydává za někoho, kým není. Lze rozlišit autentizaci osoby či autentizaci zprávy. Pro potřeby platformy se uvažuje první možnost. Při přístupu k ní je potřeba ověřit proklamovanou identitu přistupujícího uživatele. Pokud ověření dopadne ve výsledku kladně, lze konstatovat, že přistupující uživatel je tím, kdo ve skutečnosti je. I zde může samozřejmě nastat situace, kdy se podaří nějakému útočníkovi ukrást identitu skutečného uživatele.

Vzhledem k závěru z kapitoly 3.3, kde bylo konstatováno, že komponenta poskytovatele identit může být vytvořena jako webová aplikace, se nabízí následující postup. Autentizace započne ve formuláři na webové stránce, na kterou uživatel přistoupil. Po vyplnění tohoto formuláře, jehož obsahem je pole pro email a heslo, což jsou mimo jiné klíčové identifikační znaky *hlavního přístupového účtu*, se formulář odešle na server. Vyhodnocení spočívá ve vyhledání odpovídajícího *hlavního přístupového účtu* a následně ověření správnosti zadaného hesla. Pokud operace proběhne kladně, tedy je nalezen příslušný *hlavní přístupový účet* a bylo zadáno správné heslo, pak je uživatel přesměrován na další krok k získání přístupu do platformy, a to autorizaci. Ta je popsána v následující kapitole.

4.3 Autorizace

Autorizace je proces získání oprávnění k provedení operace či přístupu ke zdrojům. Také se tak označuje ono vlastní oprávnění, ať je ve formě např. náhodného řetězce určité délky, či logické hodnoty. Zpravidla se před autorizací ještě provádí proces ověření identity uživatele, který žádá o autorizaci. Tím je myšlena autentizace.

4.3.1 OAuth 2.0

V okamžiku, kdy je uživatel autentizován, přichází na řadu vytvoření oprávnění, které mu zajistí následný přístup ke zdrojům platformy. To lze učinit použitím protokolu OAuth 2.0, což je moderní autorizační protokol, který se stal de facto standardem pro zabezpečení webových služeb (viz [11]). Jeho hlavní výhoda tkví v tom, že uživatel může poskytnout klientské aplikaci přístup ke zdrojům poskytovatele, aniž by té aplikaci musel vyzradit své přístupové údaje, a tím ji poskytl prakticky neomezený přístup k jeho účtu. Dále umožňuje vymezit pravomoci jednotlivých klientských aplikací a sledovat využívání poskytnutých privilegií. Klientská aplikace se může autentizovat jak sama za sebe, tak i za jejího uživatele, který k tomu vydá explicitní souhlas. Díky tomu lze, bez rizika narušení ochrany osobních údajů, umožnit přístup k chráněným datům uživatelů, ke kterým jim dá jejich uživatel explicitní souhlas. Další výhoda spočívá v jeho snadné implementaci.

Protokol OAuth 2.0 specifikuje 4 základní role (viz [11]). Těmi jsou:

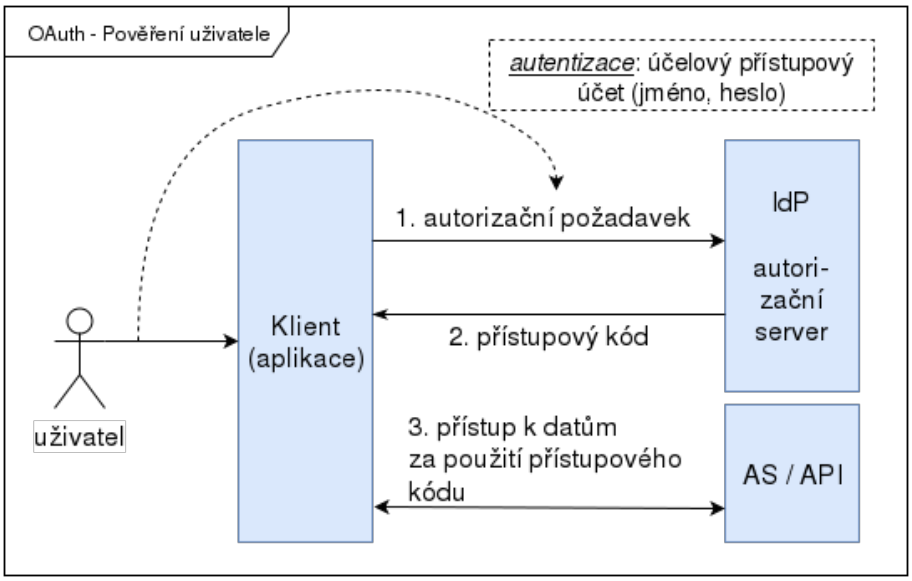
- **majitel zdroje**, který udílí přístup ke zdroji (typicky uživatel),
- **poskytovatel zdroje**, což je server, na kterém jsou zdroje uloženy a ze kterého jsou pak následně získávány,
- **klient**, což je aplikace, prostřednictvím které uživatel zadal požadavek vedoucí k získání nějakého chráněného zdroje,
- **autorizační server**, který vydává přístupové kódy klientům po úspěšné autentizaci a autorizaci.

Dále jsou definovány 4 typy udělení autorizačního oprávnění (viz [11]). Jedná se vlastně o postupy, nebo o proces, jak vydat samotné oprávnění. Jsou jimi:

- **pověření klienta**, které se používá převážně v situaci, kdy klient je zároveň majitelem zdroje; u tohoto oprávnění v rámci webových SPA vzniká problém prozrazení použitého identifikátoru klienta a hesla při autorizačním požadavku klienta, protože zdrojový kód klienta psaný v jazyce JavaScript je čitelný běžnému uživateli,
- **pověření uživatele** je nadstavbou předchozího, protože umožňuje zadat identifikátor uživatele a jeho hesla až za běhu programu, tudíž zde nehrozí nebezpečí prozrazení těchto údajů z pohledu napsaného kódu aplikace jako v předchozím případě,
- **autorizační kód** je oprávnění nejvíce používané ve spojení s tímto protokolem, jelikož postup jeho získání spočívá v několika krocích navíc oproti předchozím typům a jelikož se jedná v tomto smyslu o nejuniverzálnější řešení ze zde zmíněných pro autorizace přístupu k aplikacím třetích stran,
- **implicitní pověření**, které vychází z autorizačního kódu, ale je zjednodušeno tím, že nevyžaduje ukládání identifikátoru klienta a jeho hesla; vhodné pro webové či jiné aplikace, kde jsou volně dostupné zdrojové kódy ve kterých by byly ony údaje zjistitelné.

4.3.2 Použité způsoby autorizace

S protokolem OAuth 2.0 souvisí ještě jeden pojem, který je nutné v rámci této práce zavést. Tím pojmem je **účelový přístupový účet**. Tento typ je druhým typem přístupového účtu v rámci platformy vedle již dříve definovaného *hlavního přístupového účtu*. Smyslem tohoto druhého účtu je umožnit přístup aplikaci ke službám aplikačního serveru i za pomoci jiných přístupových údajů, než které používá *hlavní přístupový účet*. To je vhodné zejména v situaci, kdy je požadováno omezení oprávnění či výhradně účelově zřízený přístup pro konkrétní aplikaci pod identitou uživatele. Identita uživatele je pouze jedna. Ale možností přístupu může být více. Toho je dosaženo právě kvůli zavedení *účelového přístupového účtu*, jelikož počet těchto účtů v rámci jedné identity uživatele není omezen. Tento přístup je demonstrovaný na následujícím obrázku.

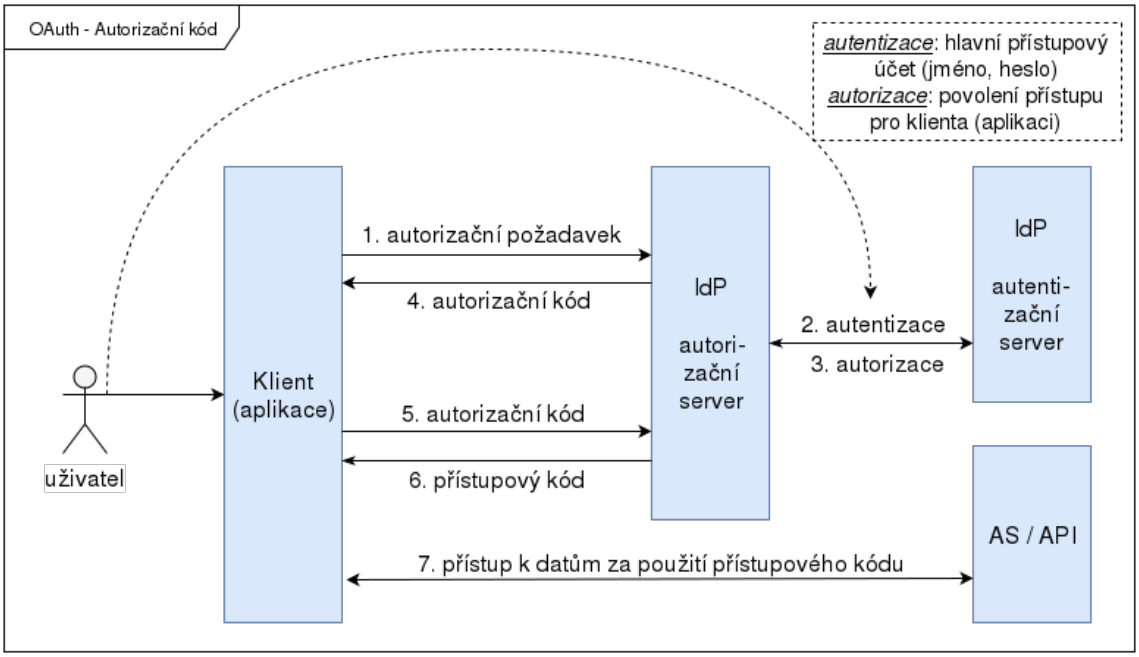


Obrázek 4.1: OAuth – postup Pověření uživatele

Na obrázku 4.1 je znázorněn postup získání přístupového kódu pomocí postupu *pověření uživatele*. Klient je v tomto případě aplikace, která potřebuje autorizovat svůj přístup, aby mohla komunikovat s aplikačním serverem. Nejprve proto odešle na autorizační server (OAuth server) autorizační požadavek. Jeho součástí jsou údaje příslušné *účelovému přístupovému účtu*, který je registrován pro použití s tímto klientem. Autorizační server odpoví na požadavek přístupovým kódem, samozřejmě pro platná data *účelového přístupového účtu* zaslaná v požadavku, který poté klient používá k přístupu na aplikační server.

Na následujícím obrázku 4.2 je znázorněn postup získání přístupového kódu pomocí postupu autorizačního kódu. Na první pohled je vidět, že se jedná o složitější proces, jelikož se tam provádí více požadavků a odpovědí. Klient nejprve odešle autorizační požadavek na autorizační server. Ten jeho požadavek přesměruje dále na autentizační server, protože daný uživatel není autentizován. Z principu funkce

samotného autorizačního serveru, kdy ten neprovádí ověření identity, musí požadavek předat dále. Autentizační server poskytovatele identit je důvěryhodnou stranou, které uživatel sdělí své údaje *hlavního přístupového účtu*. Pokud ověření proběhne korektně a uživatel je tímto autentizován, přejde se k dalšímu kroku, kterým je zobrazení žádosti o potvrzení přístupu. Pokud uživatel vyjádří souhlas, povolí tím přístup klienta k jeho datům prostřednictvím aplikačního serveru. Klientovi se pak přes autorizační server vrátí zpět autorizační kód. Tento kód slouží jednak jako potvrzení, že uživatel povolil klientovi přístup, ale také k výměně tohoto kódu za kód přístupový. Poté následuje výměna autorizačního kódu za přístupový kód mezi klientem a autorizačním serverem. Po jejím úspěšném dokončení má již klient validní přístupový kód pro přístup na aplikační server.



Obrázek 4.2: OAuth – postup Autorizačním kódem

Z předchozích dvou obrázků je zřetelně poznat, v čem se jednotlivé přístupy liší a proč jsou pro dané situace vhodné. V rámci vytvářené platformy je první postup (pověření uživatele) vhodné použít do komponent importu dat a provádění výpočtů. A to vzhledem k tomu, že tyto aplikace jsou navrženy jako terminálové a nemají přístup ke GUI autentizačního a autorizačního serveru prostřednictvím webového prohlížeče. Na druhé straně pro komponentu editor je použitelný druhý postup (autorizační kód). Je nutné si na tomto místě uvědomit, že i při použití rozdílných postupů autorizace je vše založeno na společném základu a jedná se stále o jeden protokol, který nabízí takovéto možnosti. A právě to je možné vyzdvihnout jako jeho přednost vůči jiným řešením.

4.4 Implementace

Součástí této komponenty je správa identit, autentizační server i autorizační server. Celá komponenta je postavena na frameworku Phalcon PHP ve verzi 2.0.9. Implementace správy identit byla řešena pouze do úrovně funkčního databázového modelu (viz příloha A), kdy si z této databáze autentizační server přebírá potřebné údaje o identitě a hlavním přístupovém účtu. Autentizační server je součástí komponenty a je vytvořen jako aplikace za použití modelu MVC. V této části je implentován proces ověření identity uživatele na základě zadaných údajů hlavního přístupového účtu (emailu a hesla). To vše samozřejmě graficky pomocí webového GUI.

Poslední částí této komponenty je pak autorizační server. Vzhledem ke skutečnosti, že záměrem této práce nebylo takový server tvořit, tak byl použit volně dostupný OAuth 2.0 Server PHP (viz [12]), který je šířen pod licencí MIT. Tento server samozřejmě podporuje výše rozebírané postupy, tudíž jej stačilo správně nastavit a integrovat do projektu jako službu.

Výstupem implementace je tedy webová aplikace psaná objektovým přístupem s návrhovým vzorem MVC v jazyce PHP, která je hostována na webovém serveru s podporou běhu PHP skriptů.

4.5 Testování

Testování této komponenty probíhalo za pomocí následujících technik. Po vytvoření daného kódu byl tento kód testován programátorem. Vzhledem k faktu, že se zde využíval externí projekt implementující protokol OAuth a vytvářející autorizační server, a jeho vývojáři prováděli před jeho vydáním vlastní testování, tak tato část nebyla zvláště testována. Nad celou komponentou pak proběhlo systémové testování, jenž mělo za úkol ověřit její funkčnost. To bylo provedeno autorem. Tímto testováním komponenta prošla.

5 Aplikační server

Aplikační server je stěžejní komponentou celé platformy. Tato skutečnost vychází z kapitoly 3.3, jejímž závěrem je použití aplikačního serveru jako prostředníka mezi ostatními komponentami a úložištěm informací ve formě databáze. S ohledem na zaměření této práce na webové technologie se vývoj této komponenty ubírá směrem webových služeb a s tím spojených principů a postupů.

5.1 Princip webových služeb

Webovou službu je možné definovat jako webové API, které umožňuje komunikaci systémů. V tomto momentu je důležitá právě myšlenka o komunikaci systémů, protože webové služby běžně nemají svoje GUI. Místo něj široce používají text jako formát pro výměnu dat. Komunikace pak probíhá bez ohledu na to, v jakém jazyce jsou systémy vytvořeny. Z toho je vyplývající také platformová nezávislost. Webové služby jako technologie jsou zastřešovány konsorciem W3C, které také definovalo jejich standard. Tento standart, dostupný z [13], by bylo možné volně přeložit následovně.

„Webová služba je softwarový systém podporující spolupráci mezi dvěma stroji prostřednictvím počítačové sítě. Rozhraní webové služby je popsáno pomocí strojově zpracovatelného formátu. Ostatní systémy komunikují s webovou službou v souladu s předepsaným WSDL rozhraním s použitím SOAP zpráv typicky pomocí protokolu HTTP s XML serializací, která je v souladu s dalšími webovými standardy.“ [13]

Webové služby lze rozlišit na dvě základní kategorie. První z nich jsou služby splňující podmínku stylu REST. Tyto služby využívají pro zpracování dat sadu jednotných operací, které jsou pro všechny služby stejné. Druhou hlavní třídou jsou potom libovolné služby (arbitrary web services). Tyto služby mají na rozdíl od REST služeb možnost definice libovolné množiny operací. Tato množina je popsána pomocí WSDL (Web Services Description Language). [13]

5.2 Přehled webových služeb

5.2.1 SOAP

SOAP byl vytvořen jako odlehčený protokol pro výměnu strukturovaných informací v decentralizovaném a distribuovaném prostředí. Je následovníkem protokolu XML-RPC, který taktéž sloužil pro výměnu strukturovaných informací. SOAP definuje

strukturu jednotlivých zpráv, které si aplikace v distribuovaném prostředí vyměňují. Struktura zpráv je definována ve formátu XML. Jedná se o protokol aplikační vrstvy, který však k samotnému odesílání jednotlivých zpráv používá jiný protokol aplikační vrstvy, převážně pak HTTP. [14]

5.2.2 REST

REST (Representational State Transfer) byl vyvinut jako architektonický styl distribuovaných hypermédií (jako např. World Wide Web). Vznikal souběžně s protokolem HTTP 1.1. Díky tomuto souběhu lze považovat WWW jako největší známou implementaci systému odpovídajícímu omezením REST. Samotná myšlenka REST však není svázaná s žádným protokolem. Toto tvrzení lze doložit publikovanými články, které připisují architektuře REST ocenění za to, že definuje vlastnosti, které činí WWW tak úspěšným (viz [15]).

Princip architektury REST lze demonstrovat následovně. Web se skládá z jednotlivých zdrojů. **Zdroj** je přístupný za pomoci své unikátní URL. Pokud chce uživatel přistoupit ke zdroji, zadá do prohlížeče jeho URL. Výstupem, který se uživateli zobrazí, může být HTML stránka, text či jiné audio vizuální prvky. Tím, že uživatel tento zdroj zobrazí, přechází klientská aplikace do nějakého stavu. Pokud chce uživatel následně přistoupit na jiný zdroj, zvolí některý z odkazů na onen zdroj, který dostal v předešlé odpovědi na svůj požadavek. Tím je dosaženo změny reprezentace zdroje a také změny stavu aplikace. Stav aplikace se tedy změní při provedení přístupu ke zdroji, jehož obsah se musí přenést do uživatelova prohlížeče.

Aby bylo možné definovat architektonický styl jako takový, je nutné uvést seznam omezení, která pak umožní formálně říci, zda se jedná o styl splňující požadavky REST či nikoliv. Pro tyto účely existuje šest základních omezení na systémy splňující myšlenku REST. Omezeními jsou:

- architektura klient-server,
- bezstavovost,
- možnost využití vyrovnávací paměti,
- kód na vyžádání,
- vrstvený systém,
- jednotné rozhraní.

Takový systém, který splňuje uvedená omezení, lze pak označit jako tzv. RESTful systém. Každé z těchto omezení přináší do systému své výhody i nevýhody, které ovlivňují jeho návrh a funkčnost, a s nimiž je nutné předem uvažovat. Nicméně mezi hlavní výhody tohoto stylu se řadí škálovatelnost komunikace mezi aplikacemi, její jednotná specifikace v rámci rozhraní pro komunikaci mezi aplikacemi, možnost použití prostředníků v komunikaci, robustnost a efektivita při využití vyrovnávací paměti a také snazší implementace díky existenci jednotného rozhraní. [14]

5.2.3 Srovnání

Hlavním a zcela evidentním rozdílem obou přístupů je způsob, jakým přistupují k volání vzdálených zdrojů. Metody RESTful služeb jsou omezeny existujícími HTTP metodami, naproti tomu SOAP umožňuje vytváření vlastních metod. SOAP však do jisté míry používá pouze HTTP metodu POST pro všechny operace, ať už jde o operaci získání nebo vytvoření objektu. Nevýhodou SOAP se také může zdát přidávání obálky na data ve formě SOAP zprávy. Tato obálka s sebou nese omezení typu vhodného formátu přenášené zprávy. To může být v případě jiného formátu, než XML, problém. REST služba naopak podporuje volbu formátu přímo v hlavičce HTTP požadavku. Dalším omezením, spojeným se zprávou SOAP, je množství přenesených dat pro získání nějakého zdroje. RESTful služba pro získání zdroje identifikovaného pomocí ID použije pouze URL obsahující toto ID, avšak SOAP služba musí zaslat XML zprávu dle formátu SOAP v těle HTTP požadavku. Tato zpráva pak obsahuje ID zdroje a metodu, pomocí níž lze zdroj získat. Je ale nutné vyzdvihnout rozšířenost protokolu SOAP. Ten se naopak objektově tvořícím programátorům může více zamlouvat právě z hlediska definice metod a nikoli zdrojů jako takových. Také je hojně rozšířen v prostředí podnikové integrace.

Nutno také vzpomenout, že v současné době se moderní webové aplikace navrhují výhradně stylem REST, a to z uvedených důvodů. Vzhledem k faktu, že aplikační server platformy má být poskytovatelem webových služeb, tak na základě uvedeného srovnání je za vhodnější volbu považován právě REST.

5.3 Návrh API

API, application programming interface či česky aplikační programové rozhraní, je v kontextu webových služeb takové rozhraní, které je přístupné po webu, nejčastěji pak prostřednictvím protokolu HTTP. Jeho smyslem je jasně definovat strukturu zdrojů, které daný server implementující toto rozhraní poskytuje a které tak programátor může využít. Výhodou pak samozřejmě je, že programátora nezajímá konkrétní implementace na straně serveru, který rozhraní implementuje. Jeho objektem zájmu zůstává pouze rozhraní jako takové a jeho specifikace. Vzhledem k tomu, že aplikační server je navrhován jako poskytovatel webových služeb, pak i on bude implementovat API určitých specifikovaných služeb a funkcionalit s použitím stylu REST.

5.3.1 Nástroje a jazyky specifikace

Pro definici navrhovaného API je možné využít vícero volně dostupných nástrojů a jazyků, se kterými tyto nástroje pracují. Patří mezi ně:

- **API Blueprint** (viz [16]),
- **Swagger** (viz [17]),
- **RAML** (viz [18]).

Prvně uvedený jazyk je hostovatelný ve službě Apiary.com. Pomocí ní lze online vytvářet a sdílet vyvíjená API. Druhý a třetí jazyk jsou založeny na formátu YAML, což je formát pro serializaci strukturovaných dat. Takto zapsaný kód je čitelný díky poměrně přísným požadavkům na jeho formátování. Pro oba tyto jazyky jsou dostupné i webové editory, které může uživatel hostovat i lokálně, a prostřednictvím kterých může vyvíjet své specifikace. Zajímavou možností je pak u všech nástrojů uvedených jazyků tzv. mocking service, česky simulovaná služba. Při její aktivaci jsou dané editory schopny spustit server, který simuluje právě vyvíjené rozhraní. Vývojář tak má další možnost, jak si ověřit svoje navrhované API.

5.3.2 Reprezentace zdrojů

Pro účely specifikace API aplikačního serveru byl zvolen jazyk RAML. Vzhledem k tomu, že API musí být přístupné minimálně po lokální síti, je potřeba definovat strukturu URL. Jako vhodný a obecně doporučovaný formát je uváděn následující:

https://api/v1/requested/resource?params

Místo slova *api* může být uvedena IP adresa či FQDN serveru. Za lomítkem pak následuje údaj o použité verzi. V tomto případě *v1*. Za dalším lomítkem se pak nachází cesta zdroje s potřebnými parametry.

Jako formát přenášených dat (pro požadavky i pro odpovědi) je možné zvolit z vícero nabízených, mezi které se řadí např. XML, JSON, či prostý text. S ohledem na skutečnost, jaké aplikace s ním budou pracovat, byl vybrán formát JSON. Ten je platformově nezávislý, s možností zápisu objektů, polí a samozřejmě i prostých hodnot.

Meta objekt

Každá odpověď by ideálně měla obsahovat URL, ze které je daný zdroj přístupný. Nebo-li přesně tu URL, která byla v požadavku předána serveru, podle které pak server požadavek zpracoval a vrátil odpověď. Tato URL tedy jednoznačně identifikuje zdroj. Toho lze dosáhnout za použití různých technik. Zde použitou je přidání **_meta** objektu do odpovědi, který pak obsahuje atribut **href**. Do něj se URL zdroje vkládá. Tento objekt je znázorněn na následující ukázce 5.1.

Ukázka 5.1: Meta odkaz

```
{
        "_meta": {
                "href": "https://api/v1/requested/resource",
        }
}
```

Záznam

Odpověď typu záznam odpovídá svým obsahem a zápisem jednomu konkrétnímu zdroji specifikovanému pomocí jeho jedinečné URL. Kromě samotných dat zdroje

obsahuje též **meta** objekt, jak je popsáno výše. Na ukázce 5.2 je uvedena odpověď aplikačního serveru při získání konkrétního **schematu** sítě identifikovaného pomocí jeho identifikátoru s hodnotou **1**.

Ukázka 5.2: Záznam

```
{
        "_meta": {
                "href": "https://api/v1/schemes/1",
        },
        "name": "SchemeNo1",
        "version": 1,
        "lock": true
}
```

Kolekce záznamů

Jak již název napovídá, tak odpověď typu kolekce záznamů obsahuje více jednotlivých záznamů. Ty jsou přenášeny v objektu **items**. Kromě přenášených záznamů obsahuje též **meta** objekt, a další objekty potřebné pro stránkování. Těmi jsou **first**, **previous**, **next**, **last**. Obsahem každého z těchto objektů je zase meta objekt, ale již s konkrétními parametry URL pro provedení stránkování. Strukturu odpovědi typu kolekce záznamů zachycuje ukázka 5.3. Požadavek, předcházející této zde uvedené odpovědi, požadoval seznam schemat. Vzhledem k implicitně zapnutému stránkování se omezil pouze na jeho první stránku. O technice stránkování blíže pojednává odstavec tomu věnovaný v kapitole 5.3.4.

5.3.3 Princip HATEOAS

HATEOAS, nebo-li Hypermedia as the Engine of Application State či česky hypermédia jako aplikační stav, je jeden ze základních principů REST architektury. Jeho princip spočívá v tom, že aplikace komunikuje s aplikačním serverem skrze dynamicky poskytované médium. Aplikace nepotřebuje mít žádné předchozí znalosti o tom, jak komunikovat s libovolnou částí aplikačního serveru. Tím je dosažena jistá univerzálnost, protože tímto způsobem lze dynamicky rozšiřovat funkcionalitu.

Tento princip je patrný na ukázce 5.3. Požadavek byl proveden na URL:

https://api/v1/schemes/?**pageNumber=1**

To v překladu znamená, že požadavek se dotazoval na seznam schemat. Vzhledem k zapnutému stránkování si vyžádal pouze první stránku (tučně zvýrazněno). Odpověď z ukázky 5.3 na tento požadavek obsahuje odkaz na další stránku (v tomto případě stránku č. 2), což je přesně díky principu HATEOAS. Tím je dosaženo toho, že odpověď na požadavek obsahuje de facto příkazy, co má klient udělat, aby získal jinou stránku. Obecně je tedy cílem v odpovědi popisovat kromě dat jako takových i operace, které s nimi může uživatel provádět.

Ukázka 5.3: Kolekce záznamů

```
{
        "_meta": {
                "href": "https://api/v1/schemes/?pageNumber=1",
        },
        "items": [ ... ],
        "next": {
                "_meta": {
                        "href": "https://api/v1/schemes/?pageNumber=2"
        }},
        "previous": { ... }, "first": { ... }, "last": { ... }
}
```

5.3.4 Operace nad zdroji

Vyhledávání

Vyhledávání je technika filtrování zdrojů podle zadaných parametrů. Každý identifikovaný zdroj má specifikováno, zda-li lze pro něj vyhledávání použít a případně na jaké jeho atributy. Vyhledávání je vhodné použít pouze ve spojení s odpovědí typu kolekce. Následující URL ukazuje způsob zápisu parametrů pro vyhledávání. V tomto případě se požadavek dotazuje na taková schemata, která mají atribut **verze (version)** v hodnotě 1 a **zámek (lock)** v hodnotě 0.

https://api/v1/schemes?**q=(version=1;lock=0)**

Řazení

Za pomoci řazení je možné danou sadu vyhledaných výsledků seřadit podle zadaných atributů. Řazení je použitelné pouze ve spojení s odpovědí typu kolekce. Následující URL ukazuje způsob zápisu parametrů pro řazení. V tomto případě se požadavek dotazuje na všechna existující schemata sítí. Tuto kolekci pak seřadí podle **verze (version)** vzestupně a podle **zámku (lock)** sestupně. Směry řazení (vzestupně, sestupně) lze vynechat, a poté se implicitně použije vzestupný směr.

https://api/v1/schemes?**s=(version:asc,lock:desc)**

Rozšíření pohledu

Rozšíření pohledu je jediná technika z uvedených, kterou má smysl použít ve spojení jak s odpovědí typu záznam tak i s kolekcí záznamů. Jeho smyslem je rozšířit, nebo-li expandovat, daný objekt, pokud ten je rozšiřitelný. To se může stát např. u databázového dotazu skrze více entit, nebo v případě specifického návrhu API a závislostí mezi jednotlivými zdroji. Je však nutné toto pečlivě zvážit, protože špatné použití této techniky by zbytečně zatěžovalo zpracování požadavků např. jejich neúměrnou a zbytečnou hloubkou rozšíření. Následující URL demonstruje

požadavek dotazující se na seznam všech schemat. Jednotlivé položky v objektu *items* pak budou rozšířeny o jejich vlastní obsah. Standardně by totiž tento objekt obsahoval pouze *meta* objekty odkazující na konkrétní schema.

https://api/v1/schemes?**e=(scheme)**

Stránkování

Smyslem stránkování je omezení množiny přenášených dat v odpovědi tak, aby původce požadavku nebyl zahlcen jejich velkým objemem. Proto se po aplikování vyhledávacích a řadících kritérií nalezený výsledek rozdělí do stránek s požadovanou velikostí. V odpovědi je pak navrácena pouze konkrétní požadovaná stránka. Tuto techniku je vhodné použít pouze ve spojení s odpovědí typu kolekce. Následující URL demonstruje způsob výběru stránky pomocí parametru **pageNumber** a její velikosti pomocí **pageSize**.

https://api/v1/schemes?**pageSize=25**&**pageNumber=1**

5.3.5 Identifikované zdroje

Pro funkcionalitu platformy byly identifikovány následující zdroje nejvyšší úrovně: **schema** (scheme), **uzlový bod** (node), **úsek** (section), **mapový bod** (map point), **objekt** (object), **oprávnění** (permission), **úloha** (task), **vykonavatel** (executor) a **uživatel** (user). Jejich přesné uspořádání, použité techniky, formáty odpovědí atd. jsou předmětem provedené specifikace a jsou popsány příslušným RAML souborem včetně ukázkových odpovědí formátovaných pomocí JSON.

5.4 Implementace

5.4.1 Výběr frameworku

Za předpokladu, že jako programovací jazyk je vybrán PHP, tak pro danou implementaci je možné zvolit z nespočtu různých webových frameworků. Mezi často používané lze zařadit např. Zend, Symphony, Nette či CakePHP. Mezi výhody použití jakéhokoli frameworku lze uvést množství připravených komponent, technologií, postupů či komunitu věnující se tomu danému frameworku.

Na poli dostupných PHP frameworků se však vyskytuje jeden, který je koncipován odlišně od předchozích. Tímto frameworkem je Phalcon (viz [19]). Přestože se jedná o poměrně mladý framework, jehož počátky sahají teprve do roku 2012, tak jeho šíři a možnosti lze porovnávat s předešlými. Stěžejním rozdílem je způsob jeho použitelnosti, resp. distribuovatelnosti. Zatímco běžné frameworky se distribuují jako samostatné PHP projekty, které interpret jazyka PHP musí za běhu interpretovat, tak Phalcon je distribuován jako přímé rozšíření PHP a připojuje se tak již při samém startu interpreta PHP. Interpret tak následně interpretuje pouze vytvořenou aplikaci a framework, jako takový, již nikoli. S tím také souvisí skutečnost, že Phalcon je interně kompilován do jazyka C. Tato kompilace z něj vytvoří rozšíření pro

PHP, které pak lze k interpretu PHP připojit. A právě jeho existence je důvodem, proč byl pro účely tvorby aplikačního serveru vybrán.

5.4.2 Architektura

Aplikační server je založen na architektuře MVC, což je softwarová architektura oddělující datový model, uživatelské rozhraní a řídící logiku. V tomto okamžiku je nutné uvést, že veškeré odpovědi serveru jsou realizovány pomocí textového výstupu formátovaného prostřednictvím JSON. To znamená, že uživatelské rozhraní figuruje pouze do úrovně formátování výstupu, přičemž výstupem je prakticky strojový formát dat a nikoli GUI. To je ale přesně ta funkcionalita, která je od aplikačního serveru v kontextu platformy očekávána.

Každý zdroj, který byl identifikován v kapitole 5.3.5, zde vytváří vlastní kontrolér (controller). Ten je zodpovědný za veškerou řídící logiku pro jednotlivé zdroje v něm implementované. Všechny takto vzniklé kontroléry mají společného předka tzv. *ControllerBase*, který nad nimi zajišťuje ověření přístupu pro každý příchozí požadavek. Vzhledem k tomu, že všechny zdroje jsou přístupné pouze autorizovaným uživatelům, resp. jejich požadavkům s validním přístupovým kódem, tak tento jejich společný předek právě toto ověření provádí. Ověření spočívá v tom, že se hledá v databázi **OAuth** a entitě **přístupových kódů** (access_token) takový přístupový kód, který existuje a který je v čase přijetí požadavku časově platný. V opačném případě je pak vygenerována odpověď s návratovým chybovým kódem 403, který oznamuje neautorizovaný přístup ke zdroji.

Pro účely splnění požadavku z kapitoly 3.1, který požadoval omezení přístupu na úrovni schematu, byla vytvořena komponenta ACL. Jejím účelem je při každém přístupu ke schematu ověřit, zda-li přistupující uživatel se svojí identitou má takové oprávnění, které by mu toto dovolovalo. Pro rozlišení jednotlivých druhů oprávnění k přístupu ke schematu byla zvolena textová reprezentace ve formátu **RWX**, kdy **R** označuje oprávnění ke čtení, **W** pak oprávnění zápisu a **X** oprávnění k zadání a úpravě výpočetní úlohy. Oprávnění je vázáno k identitě uživatele a příslušnému schematu. Uchovává se v databázi **sítě** (network) v entitě **oprávnění** (permission).

Díky širokému rozsahu funkčnosti pracuje tato implementace s modely z více databází, konkrétně **Síť** (viz příloha C), **Poskytovatel identit** (viz příloha A), **OAuth** (viz příloha B) a také **Výpočetní úlohy** (viz příloha D). Dále je zde použita technika ORM, což v překladu znamená objektově relační mapování. To spočívá v mapování objektů, v tomto případě modelů aplikace, na jejich ekvivalenty v databázi.

5.4.3 Zabezpečení

I v rámci vývoje webových služeb, resp. aplikačního serveru poskytujícího webové služby, je nutné uvažovat potenciální zranitelnosti, které s tím souvisejí. Mezi základní typy těchto zranitelností patří XSS (viz [20]), SQL Injection (viz [21]) a CSRF (viz [22]). K ošetření těchto zranitelností existují doporučené postupy, které minimalizují možnost provedení útoků.

Zranitelnosti XSS a SQL Injection jsou si do jisté míry podobné. Ke svému účelu potřebují modifikovaný vstup od uživatele. Jejich rozdílem je pak místo, kam se škodlivý kód předává a co má způsobit. Zranitelnost typu XSS cílí na JavaScriptový kód, kdy v případě nevhodně ošetřených vstupů může být zadaný kód proveden. Pokud k takovému provedení dojde, existuje riziko zneužití vlastních dat. Naopak zranitelnost typu SQL Injection využívá upravený vstup uživatele k získání dat z databáze, kdy lze pomocí špatně ošetřených vstupů sestavovat libovolné dotazy, které budou následně nad databází provedeny. Mezi základní techniky ošetření vstupů se používají regulární výrazy, odstranění bílých znaků po stranách řetězce a escapovaní znaků. Též se využívají tzv. připravené dotazy (prepared statements), které předchozí techniky korektně implementují v rámci zvoleného frameworku. Aplikační server využívá připravené dotazy pro komunikaci s databázemi, takže výše uvedená rizika jsou minimalizována.

Další zranitelností je CSRF, což je útok spočívající v tom, že uživatel je donucen navštívit stránku, která skrytě vykoná útok na webovou aplikaci, kde je uživatel zrovna přihlášen. Ošetřením této zranitelnosti je generování jedinečného kódu pro každou akci, která svým provedením mění data. Před provedením akce se pak zkontroluje poslaný a předem vygenerovaný kód. Tyto kódy se musí shodovat. V opačném případě je možné, že se jedná o útok. Zde je vhodné vzpomenout, že zde existuje více možností, jak toto zabezpečení realizovat. Nejdříve je nutné si uvědomit, že v rámci požadovaného zabezpečení přístupu k aplikačnímu serveru je nutný validní přístupový kód. Ten se vytváří na základě úspěšné autentizace a autorizace, přičemž je tvaru náhodného řetězce určité délky. Už to samo o sobě je určitou mírou zabezpečení, jelikož bez jeho existence bude požadavek vždy zamítnut. Využívá se také jako sdílený kód právě pro potřeby CSRF. Vylepšením tohoto postupu by bylo generování dalšího kódu pro potřeby CSRF, který by byl svázán s přístupovým kódem. To by bylo výhodné zejména kvůli faktu, že by existovaly dva rozdílné kódy a bylo by tak obecně složitější je zjistit. Posledním vylepšením by bylo předešlý kód pro CSRF aktualizovat při každém přijetí požadavku měnícího stav dat. Po jeho provedení by byl v odpovědi zaslán nově vygenerovaný kód, který by sloužil pro ověření dalšího požadavku.

5.5 Testování

Testování této komponenty probíhalo za pomocí následujících technik. Po vytvoření kódu byl kód testován programátorem. Vzhledem k existenci objektů, u kterých mělo smysl je podrobit jednotkovému testování, byly tyto objekty testovány za pomocí PHPUnit, což je knihovna pro psaní jednotkových testů, kterou lze využít i v případě projektu založeném na frameworku Phalcon. Nad celou komponentou pak proběhlo systémové testování, jenž mělo za úkol ověřit její funkčnost. To bylo provedeno za pomoci nástroje Codeception, což je nástroj poskytující více kategorií testů. Pro testování na úrovni systémového byl zvolen modul pro testování REST API. Pro něj byla vytvořena sada scénářů, které testovaly práci s jednotlivými implementovanými zdroji. Uvedeným testováním komponenta prošla.

5.6 Provozní prostředí

Aplikační server je napsán ve skriptovacím jazyce PHP a tím pádem je pro něj potřebné zajistit interpret, který jej provádí. Tím je PHP-FPM, který se spouští z webového serveru. Jako webový server je zvolen Nginx, díky svému přehlednému a snadnému nastavení a také z hlediska výkonu. Samozřejmě je možné zvolit i jinou kombinaci. V tom případě bude ale nutné upravit pravidla pro přepisování URL, protože aplikační server obsahuje v tento moment pouze pravidla pro použití s Nginx serverem.

Kvůli zajištění bezpečné komunikace mezi aplikačním serverem a ostatními komponentami jsou jeho webové služby dostupné pouze pomocí HTTPS protokolu, což je nadstavba síťového protokolu TCP, která umožňuje zabezpečit komunikaci mezi klientem a serverem pomocí šifrování. Ke správné funkci šifrování je zapotřebí, aby server disponoval platným certifikátem, kterým je komunikace šifrována. Za zmínku stojí možnost získání validního certifikátu od certifikační autority Let's Encrypt, která na sklonku roku 2015 začala takovéto nekomerční certifikáty vydávat, aby podpořila šifrovaný provoz na webu. S použitím šifrovaného spojení je vhodné ještě nastavit HSTS hlavičku (viz [23]), která prohlížeči říká, zda má používat šifrované spojení pro daný přístup k serveru. Rovněž je možné v aktuální verzi Nginx aktivovat podporu protokolu HTTP2 (viz [24]), který poskytuje určité výhody z hlediska výkonu. To je dáno tím, že se jedná o binární protokol, který podporuje multiplexování požadavků a který řeší i komprimaci přenášených hlaviček. Na serveru je také nutné nastavit CORS (viz [25]), aby byla zajištěna podpora v případě požadavků z jiné domény.

5.7 API Connector

API Connector je desktopová aplikace vytvořená v jazyce Java. Implementuje spojení s aplikačním serverem a odstiňuje dále vyvíjené aplikace od potřeby vlastní implementace API poskytovaném aplikačním serverem. Jde tak v podstatě o wrapper, nebo-li obal, k API aplikačního serveru na lokální úrovni. Z kapitoly 3.1 je v tuto chvíli zřejmé, že minimálně komponenta importu a komponenta výpočtů mohou být realizovány např. v jazyce Java. Budou přímo komunikovat s aplikačním serverem prostřednictvím jím poskytovaného API. Proto je v tento moment vhodné takovýto wrapper připravit a následně při jejich tvorbách pouze využít. Při znalosti komponent, které tento konektor využijí, jsou pro jejich potřeby implementovány pouze takové zdroje (v balíčku *resources.**), které se prakticky využijí. Následně je možné v budoucnu doimplementovat zbývající zdroje podle potřeby. Implementované zdroje byly jednotkově otestovány. Těmito testy všechny zdroje prošly.

6 Editor

Účelem této komponenty je poskytnout běžnému uživateli grafický nástroj pro práci se schematem sítě, prvky schematu a také s tím související provádění dostupných typů výpočetních úloh. Doposud zmíněné a popsané komponenty tuto funkcionalitu totiž nenabízejí a z pohledu editoru jsou podpůrné, avšak nezbytné, pro jeho činnost. Jedním ze závěrů kapitoly 3.3 je skutečnost, že výpočty dostupných výpočetních úloh jsou prováděny na samostatném uzlu k tomu určeném. Z toho vyplývá, že editor má podporovat správu takovýchto výpočetních úloh, nikoli je aktivně provádět. K editoru má být umožněn přístup pouze autorizovanému uživateli. Autorizaci a jí předcházející autentizaci řeší poskytoval identit (viz kapitola 4).

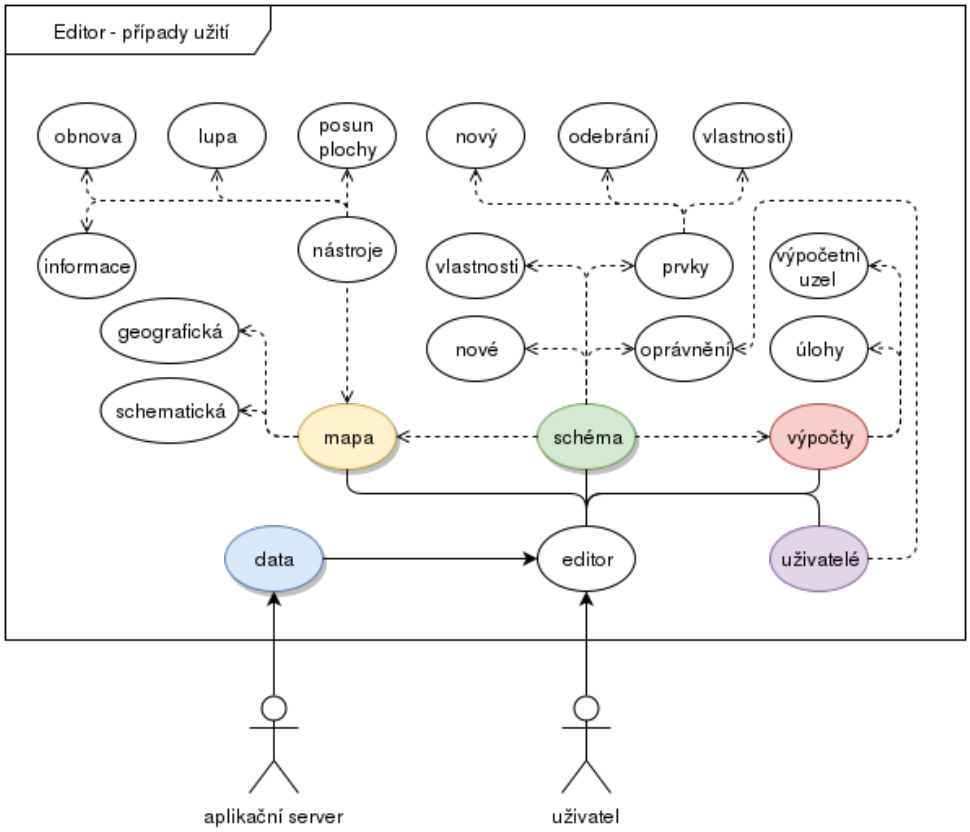
6.1 Návrh aplikace

Základním kritériem webové aplikace je, kromě jejího provozu v rámci webu, způsob jejího spuštění a následného zpracování požadavků. To může probíhat odesláním požadavku na server a opětovným stažením nově vygenerované stránky. Naproti tomu je možné stránku načíst pouze jednou a požadavky pak předávat na server odděleně a pouze zpracovávat přijaté odpovědi. To vše tedy bez znovu načítání nově vygenerované stránky. Tuto druhou techniku je možno nalézt pod zkratkou SPA, nebo-li single page application či česky jednostránková aplikace. Je zřejmé, že každá z těchto technik má své výhody a nevýhody. Jedním z nich je například použití JavaScriptu, jako programovacího jazyka na straně klienta (webového prohlížeče). Zatímco prvním způsobem zpracovaná aplikace jej ke své funkci v zásadě nepotřebuje, čímž se samozřejmě neříká, že by jej nemohla využít, tak aplikace vytvořená druhým způsobem je na něm zcela závislá. Právě závislost na Javacriptu může být slabinou dané aplikace, jelikož jeho deaktivaci může ve svém prohlížeči provést samotný uživatel. V tom případě by druhá aplikace přestala fungovat. Je korektní zmínit, že právě zmíněný druhý způsob vytváření webových aplikací je trendem poslední doby a zcela jistě má své opodstatněné zastoupení.

Přístup SPA je vhodným kandidátem pro komponentu editoru. To především z důvodu potřeby dynamického vykreslování schemat, které by prvním uvedeným přístupem bylo značně komplikované. Další výhodou pak je předpokládaná úspora přenesených dat. Vzhledem k použití protokolu HTTP2 a techniky SPA, je pak tento předpoklad oprávněný. Při prvním načtení stránky se načtou všechny potřebné soubory se zdrojovými kódy a grafickými podklady, které jsou potřebné pro běh samotné aplikace, a poté již probíhá pouze komunikace s aplikačním serverem za

účelem výměny konkrétních dat konkrétních zdrojů. Tedy takových zdrojů, které aplikace potřebuje pro svůj chod.

6.1.1 Případy užití



Obrázek 6.1: Diagram případů užití

Obrázek 6.1 zachycuje diagram případů užití této komponenty. Je z něj patrných několik skutečností. Předně editor obsahuje čtyři tematické okruhy – schema, mapu, výpočty a uživatele. V rámci každého tohoto okruhu je definována jeho určitá funkcionalita.

Prvním takovým okruhem je **schema**. Schema je zastřešující prvek elektrické resp. modelované sítě. S ním jsou totiž svázány všechny prvky, které se v daném schematu vyskytují. V rámci platformy a tím i příslušného datového úložiště nemůže existovat prvek sítě, který by nebyl přiřazen do nějakého schematu. Nad každým prvkem by dále měla být funkcionalita pro jeho vytvoření, úpravy a odebrání. Se schematem ještě úzce souvisejí oprávnění. Ke schematu je totiž umožněn přístup těm uživatelům, kteří jsou u daného schematu uvedeni jako povolení a mají příslušné oprávnění. Tím je myšleno oprávnění pro čtení, zápis nebo spouštění úloh. To vše je časově omezené v rámci oprávnění.

Dalším okruhem jsou **mapy** využívající nástroje, které zahrnují operace přiblížení

a oddálení mapy, obnovení již vykreslené mapy a posun mapy podle uživatelských preferencí. Tyto nástroje by měly sloužit pro základní vizualizaci dané mapy, resp. graficky vyjádřeného schematu. Zde se dále nabízí rozšíření o další vybrané nástroje řešící např. online editaci, pokročilejší zobrazení rozsáhlých map či geografické mapy. S nimi je v současném návrhu a uspořádání datových modelů již uvažováno.

Třetím okruhem jsou **výpočty**. Jak již bylo uvedeno v úvodu této kapitoly, editor ze své podstaty výpočty nevykonává. Místo toho řeší správu úloh, které se následně předají výpočetním uzlům k provedení. Pro tyto účely tedy editor musí podporovat práci s úlohami, znát implementované výpočetní úlohy a také mít k dispozici přehled výpočetních uzlů.

Posledním okruhem jsou pak **uživatelé**. V kontextu editoru je tím myšleno pouhé zobrazení jejich seznamu, protože další funkce na toto nejsou požadovány. Zde je vhodné připomenout skutečnost, že vlastní správu uživatelských identit, a tím pádem i dostupných uživatelů, provádí komponenta poskytovatele identit platformy (viz kapitola 4).

6.2 Implementace

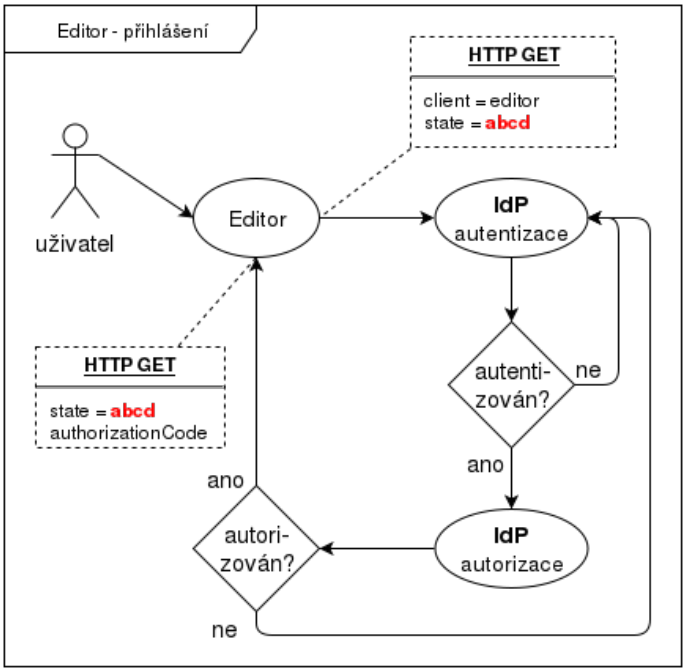
Pro tvorbu byl zvolen jazyk JavaScript a to z důvodu, že se v konečné fázi jedná o jediný jazyk pro vytváření webových SPA. Stejně jako u předešlých komponent, kde se využíval konkrétní framework, tak i zde byl takový použit, konkrétně AngularJS ve verzi 1.5 (viz [26]). Ten byl zvolen čistě na základě faktu, že v době volby se jednalo o poslední stabilní a produkční verzi. Původně bylo zvažováno použití frameworku Angular 2, nicméně kvůli probíhajícímu vývoji, a tudíž neexistence stabilní a produkční verze, od něj bylo upuštěno. Nicméně pro budoucí aktualizaci této komponenty je možné jej uvažovat.

AngularJS pomocí architektury MVC umožňuje oddělení aplikační logiky od datového modelu a uživatelského rozhraní. Už tento směr předznamenává způsob práce v takovémto projektu. Kromě architektury MVC obsahuje i další praktické techniky známé z jiných frameworků a jazyků. Mezi zásadní patří DI (Dependency Injection či česky vkládání závislostí) či Two Way Data-Binding (česky obousměrné mapování dat). Právě druhá technika je na tomto frameworku velmi praktická. Pomocí ní lze deklarativně provázat GUI s modelem. Není třeba specifikovat způsob, jakým se má GUI s modelem synchronizovat. O to se totiž automaticky postará Angular.

6.2.1 Autentizace a autorizace

Za použití Angularu byl vytvořen editor, který implementuje funkce popsané v kapitole 6.1. Vzhledem ke skutečnosti, že k datům je povolen přístup pouze autorizovaným uživatelům, tak je zapotřebí se jí blíže zabývat. Autentizaci a autorizaci řeší komponenta poskytovatele identit (viz kapitola 4). Je proto nemyslitelné, aby se cokoli z jeho problematiky odehrávalo na straně editoru. Principiálně jde totiž o situaci, kdy uživatel má zadávat své přístupové údaje hlavního přístupového účtu pouze vůči poskytovateli identit a nikoli vůči jiným aplikacím. Tím je totiž zajištěna

nezanedbatelná úroveň bezpečnosti, protože nemůže dojít k bezprostřednímu zneužití údajů ze strany jiné aplikace. Řešením je přesměrování uživatele na autentizační stránku poskytovatele identit. Ten ověří přístupové údaje. V kladném případě zobrazí uživateli autorizační formulář. Jeho obsahem je stručné informování uživatele o tom, že aplikace, ze které vzešel požadavek přístupu, se pokouší tento přístup k chráněným zdrojům získat. Pokud jej uživatel odsouhlasí, je přesměrován zpět do editoru, kde proběhne dokončení procesu přihlášení. Následně může uživatel aplikaci používat. Tento postup schematicky znázorňuje obrázek 6.2.



Obrázek 6.2: Přihlášení do editoru pomocí IdP

Z hlediska bezpečnosti je důležité k obrázku 6.2 poznamenat, že takto uvedený postup by byl sám o sobě zranitelný. Zranitelnost v tomto případě spočívá v CSRF (viz [22]). V okamžiku příchozího autorizačního kódu do editoru od poskytovatele identit by totiž nebylo zajištěno, že ten není nastrčen případným útočníkem, který se pokouší získat uživatelovu identitu. Proto se jako opatření použije další parametr, zde konkrétně *state*, v URL, který obsahuje pouze náhodně generovaný řetězec. Při příjmu odpovědi od poskytovatele identit se pak porovnává takto poslaný a lokálně uložený řetězec s dorazivším. Pokud ten chybí či neodpovídá, jedná se pravděpodobně o útok. V tom případě se přihlášení nedokončí. Dokončením přihlášení se rozumí výměna dorazivšího autorizačního kódu za kód přístupový, která se následně odehraje mezi klientem a poskytovatelem identit.

Aby se nemusel při každém posílaném požadavku na aplikační server do hlaviček programově doplňovat přístupový kód, je zde využita technika tzv. **Interceptor**. Jedná se de facto o službu, která se provede před odesláním požadavku či při přijetí odpovědi. Samozřejmá je i možnost jejich použití s chybovou odpovědí. Zde v editoru

je implementována první funkcionalita, tedy před každým voláním požadavku na aplikační server se do tohoto požadavku vloží autorizační hlavička s přístupovým kódem. Pro případ příchodu jakékoli odpovědi od aplikačního serveru, která by obsahovala návratový kód 401 nebo 403, což by značilo neplatný přístupový kód, se zde implementačně ošetřuje i dorazivší odpověď.

Pro účely grafické vizualizace schematu elektrické sítě je použita externí knihovna **vis.js**, která se zaměřuje na tvorbu grafů, sítí, časový řad atp. Její ovládání řeší příslušná komponenta editoru, která vykresluje schematickou mapu zvolené sítě, resp. schematu. Dále byly použity vybrané konstrukty z nového ES2015, což je nová verze ECMAScriptu. Mezi tyto konstrukty patří např. třídy v obvyklém významu, blokově viditelné proměnné, moduly a tzv. **promisses** což jsou de facto třídy, jejichž hodnota bude známá v budoucnu. Ty se používají při asynchronním programování a jejich praktické uplatnění je při komunikaci s externími zdroji, v případě editoru komunikace s aplikačním serverem.

Celý kód komponenty je vytvořen v prostření **node.js** za použití JavaScriptového kompilátoru **Babel**, správce balíků **Bower** a balíčkovacího systému **Webpack**. Ten, jako finální úpravu, provádí sloučení všech souborů s programovými částmi do jediného, což ve výsledku šetří počet HTTP požadavků.

V příloze E je pro ukázku zobrazena schematická mapa části schematu sítě Želkovice.

6.3 Testování

Testování této komponenty probíhalo za pomocí následujících technik. Po vytvoření daného kódu byl kód testován programátorem. Vzhledem k faktu, že se zde využívaly externí knihovny a moduly, které před svým zveřejněním prošly vlastním testováním, tak tato část nebyla zvláště testována. Nad celou komponentou proběhlo systémové testování, jenž mělo za úkol ověřit její funkčnost. To bylo provedeno autorem. Tímto testováním komponenta prošla.

6.4 Provozní prostředí

Editor je napsán v jazyce JavaScript za použití HTML, CSS a dalších potřebných technik. S tím souvisí i možnosti jeho provozování. Jelikož se jedná o klientskou aplikaci (SPA), která se tedy bude spouštět na straně uživatele v jeho webovém prohlížeči, pak není potřebné na straně serveru používat jakékoli skriptovací jazyky. Webový server tak má pouze za úkol distribuovat statický obsah. Tím jsou myšleny právě JavaScriptové kódy, HTML, styly atp. Jako webový server byl v tomto případě zvolen Nginx. Nic ale nebrání použití kteréhokoli jiného. Ve spojení s Nginx, a také s důrazem na bezpečnost, je nastaveno zabezpečené spojení HTTPS mezi tímto distribučním serverem a webovým prohlížečem uživatele. Stejně jako u aplikačního serveru, tak i tento používá certifikát vydaný certifikační autoritou Let's Encrypt a také HTTP2 protokol.

7 Propojení se systémy třetích stran

Jak již název samotné kapitoly naznačuje, je na místě v tento moment uvažovat ještě o možnosti propojení platformy se systémy třetích stran. Stačí vzpomenout na úvod kapitoly 2, ve kterém je zmíněna přítomnost systému RIS u konzultanta. Minimálně z tohoto úhlu pohledu by se tedy mohlo jednat o vhodnou komponentu platformy, protože je pravděpodobné, že potenciální uživatel této platformy bude obdobný software již používat. Vzhledem k proklamované otevřenosti platformy by tak mohl i tento uživatel projevit zájem o propojení systému s platformou, pokud by již neexistovalo.

7.1 Principy propojení systémů

Dva obecně heterogenní systémy nemusí být vůbec snadné propojit. Zvláště, pokud se jedná o komerční produkty, ke kterým není přístup ve formě zdrojových kódů či přístup k datovým úložištím. Tato situace má pak alespoň dvě strany. Jednou je systém, který data poskytuje. Druhou je pak systém, který data přijímá a ukládá. Minimálně na úrovni praktických možností, a zdaleka nepostihující všechny možnosti, je následující výčet. V něm je pod zkratkou DB myšlena databáze a pod zkratkou API libovolné API např. ve formě webových služeb. Některé z možností propojení jsou:

- **DB – DB**,
- **API – API**,
- **soubor – soubor**,
- **soubor – DB**,
- **soubor – API**.

Databáze, při vhodném hardwarovém vybavení a při jejím vhodném návrhu a optimalizaci, může vynikat svými výkony na poli rychlosti zpracování a organizaci dat. Její nevýhodou je softwarová závislost na konkrétně zvolených databázích a v případě jejich nehomogenity, myšleno neexistence totožných databázových serverů na obou stranách, i vzájemná nekompatibilita. To lze do jisté míry vyřešit modelem s použitím souborů. Nicméně zde také vyvstává celá řada otázek počínaje způsobem přenosu, kódováním, jeho velikostí a konče jejich správou. Poslední možností je

použití API. To do jisté míry dokáže eliminovat nedostatky předchozích. Při jeho správném návrhu bude sloužit univerzálně pro více účelů a tím lze i minimalizovat softwarové náklady. Nelze tvrdit, že jakákoli možnost je špatná. Vždy bude záležet na konkrétní situaci a parametrech, které do ní budou vstupovat. Jako příklad parametrů lze uvést např. spojení po síti a s tím související rychlost či prodleva přenosu, velikost přenášených dat atp.

7.2 Import dat

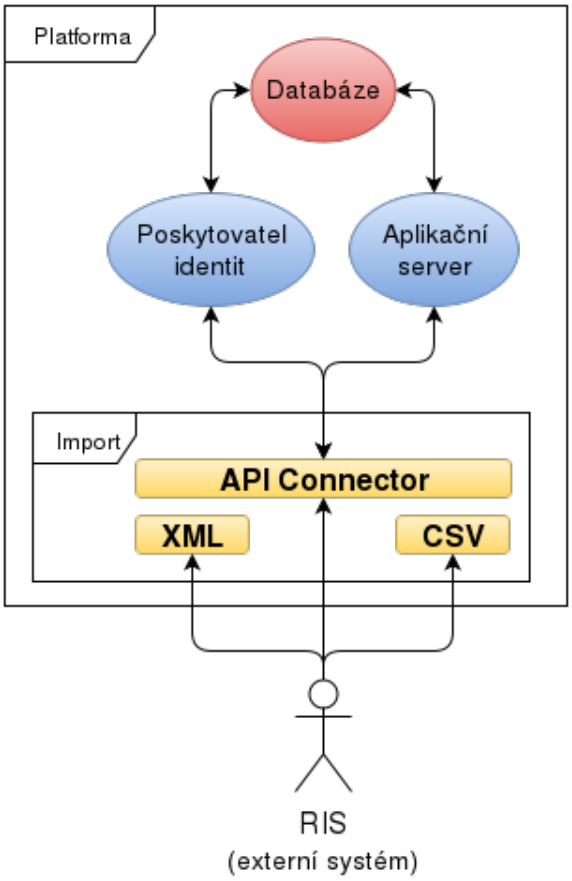
První logickou částí této komponenty je import dat z externího systému do platformy. Pro tyto účely je nadále za onen externí systém považován systém RIS. V tento moment je vhodné rozdělit import do dvou pohledů.

Prvním pohledem je systém, který data bude přijímat, tedy platforma. Existují dva smysluplné způsoby, jak přijímaná data zpracovávat a uchovávat. Tyto závisí na modelech platformy, které byly uvedeny v kapitole 3.3. V případě platformy založené na centrálním úložišti dat (viz obrázek 3.1) by jejím centrálním prvkem byl databázový server. Mělo by tak smysl provádět import vůči platformě přímo do její databáze. Což samo o sobě nelze považovat za špatnou techniku. A vzhledem k tomu, že ostatní komponenty v tomto modelu s databází spolupracují, je to jeden z praktických způsobů, jak zajistit dostupnost importovaných dat pro ostatní komponenty a jak import provést. Platforma je navržena podle modelu s aplikačním serverem (viz obrázek 3.2), který obsahuje jako centrální prvek aplikační server. Ten je prostředníkem mezi databází, jako datovým úložištěm, a ostatními komponentami. Přestože i v tomto případě by přímému provedení importu do databáze nic nebránilo, jedná se spíše o teoretickou možnost. Tím by totiž byla ztracena jistá míra abstrakce, kterou aplikační server pro ostatní komponenty nad databází vytváří pomocí svého API.

Druhým pohledem je zcela analogicky systém, který data pro import poskytuje. Nebo-li ten, ze kterého se data převádějí do platformy. V tomto případě tedy RIS. Zde je situace o poznání komplikovanější. To je ze své podstaty zapříčiněno jeho komerční povahou. Pro začátek je možné konstatovat, že platforma je vybudována tak, aby ji mohli využít i jeho tvůrci. Tím je myšlena možnost implementace API aplikačního serveru platformy vývojáři systému RIS. Samozřejmě v případě jejich vůle. Z uvedeného je však opět patrná závislost na tvůrci systému RIS. Nicméně za pomoci konzultací u konzultanta byla zjištěna ještě jiná možnost, jakým způsobem je možné import provést. Konzultant běžně pracuje i se soubory formátu CSV, kde jsou specificky uloženy atributy daného schematu. Data těchto souborů jsou přebírána ze systému RIS a obsahují potřebné parametry, které jsou požadovány i platformou. V tomto okamžiku je vhodné zmínit, že tento CSV soubor nereprezentuje interní úložiště dat v systému RIS, ale jedná se o jakýsi přenosový prostředek s vybraným obsahem. Je tedy patrné, že tento způsob je v době tvorby této komponenty platformy tím jediným dostupným, a proto je použit k provedení importu dat. Jedná se tak o způsob importu **soubor – API**.

Následující obrázek 7.1 přibližuje jakým způsobem je program pro import dat

navržen. Pro potřeby autorizace a získání přístupového kódu komunikuje s poskytovatelem identit. Data jsou pak posílána na aplikační server prostřednictvím jeho API. Dále je zde patrná skutečnost jakým způsobem formátovaná data dokáže komponenta zpracovávat. Tím jsou přenosové formáty CSV a XML, které jsou popsány níže v následující kapitole. Také je zde znázorněna možnost přímého napojení API komponenty na externí systém. Avšak toto je zde pouze demonstrováno. Vzhledem k předchozímu odstavci to není implementováno.



Obrázek 7.1: Import – návrh komponenty

7.2.1 Předávací formát

Použití předávacího formátu je zvoleno z důvodu uvedených v předchozí kapitole. Principiálně by nebyl potřeba v případě přímé implementace API platformy na straně tvůrců jednotlivých systémů. Na druhou stranu je to ale vhodný prostředník pro popis daného schematu.

CSV

Tento formát je převzat od konzultanta. Jeho struktura je ovlivněna samotným způsobem formátování CSV, což jsou jednotlivé hodnoty oddělené středníkem. První

řádek obsahuje hlavičku schematu. V ní jsou uvedeny např. jeho jméno či popis. Následují dvě sekce. První z nich popisuje uzlové body, kde každý uzlový bod je reprezentován jedním řádek. V něm jsou uloženy jeho atributy např. popis bodu, jeho identifikátor v rámci tohoto souboru, souřadnice na schematické mapě a hodnoty potřebné pro výpočty. Druhá sekce uchovává úseky. Její struktura je podobná. Obsahuje např. popis úseku, jeho identifikátor v rámci tohoto souboru a další hodnoty vztažené k výpočtům. Tento formát je plně převoditelný na následující XML formát.

XML

Tento formát byl vytvořen pro potřeby platformy již v rámci magisterského projektu (viz [1]). Vychází z jejich potřeb. Je také plně převoditelný na předchozí předávací formát CSV.

7.2.2 Implementace

Na základě požadavků na funkčnost z kapitoly 3.1 byl pro tvorbu této komponenty zvolen jazyk Java. V rámci implementace je použit konektor na aplikační server, který byl blíže popsán v kapitole 5.7.

Pro autentizaci tato komponenta používá autorizační proces *pověření uživatele*. To je z toho důvodu, že v tomto případě nelze využít standardní proces *autorizační kód*, protože tato komponenta je navržena pro provoz v příkazovém řádku. Tím pádem neumožňuje provést manuální autorizaci, která probíhá ve webovém prostředí poskytovatele identit. Právě i z tohoto důvodu se zde pro přístup používají údaje *účelového přístupového účtu*. Ten může být zřízen pouze pro potřeby této komponenty, je přiřazen k identitě uživatele, který jej vytvořil, ale zároveň nemůže dojít k ovlivnění či prozrazení údajů jeho *hlavního přístupového účtu*.

Tato komponenta je navržena pro spuštění v terminálu. Jejími parametry je volba předávacího formátu a následně samotný soubor připravený k importu. Po jejich zadání komponenta provede načtení souboru, vytvoření prvků prostřednictvím objektů daného schematu. Následuje vytvoření nového schematu na aplikačním serveru, oprávnění pro aktuálního uživatele a dále jsou postupně odesílány do nově vytvořeného schematu připravené prvky. Nejprve se odesílají mapové body, poté uzlové body a závěrem úseky. Toto pořadí je určeno prostřednictvím vazeb mezi objekty. Nelze totiž např. vytvořit úsek, jehož parametrem jsou uzlové body, které ještě nejsou na aplikačním serveru vytvořeny.

7.2.3 Testování

Testování této komponenty probíhalo za pomocí následujících technik. Po vytvoření daného kódu byl kód testován programátorem. Vzhledem k faktu, že se zde využíval i API Connector z kapitoly 5.7, který byl samostatně testován, tak tato část nebyla zvláště testována. Nad celou komponentou pak proběhlo systémové testování, jenž

mělo za úkol ověřit její funkčnost. To bylo provedeno autorem. Tímto testováním komponenta prošla.

7.2.4 Reálné sítě

Po dohodě vydal konzultant souhlas s provedením importu schemat NN sítě v obci Želkovice a části VN sítě ve městě Dobrušce za účelem ověření funkcionality platformy. Schema Želkovice bylo importováno z předávacího formátu CSV. Pro druhé schema Dobruška byl vytvořen předávací formát XML, který byl importován. Tato změna formátů proběhla za účelem ověření funkčnosti XML formátu. Oba importy proběhly úspěšně. Následně byla prostřednictvím editoru provedena vizuální kontrola úplnosti schemat. V příloze E je uvedena ukázka importované sítě obce Želkovice v prostředí editoru.

7.3 Export dat

Exportem se rozumí přenos dat z platformy do externího systému. Tato funkcionalita nebyla vznešena za požadavek a není implementována. Faktem však zůstává, že platforma je na takovouto funkcionalitu připravena, a to prostřednictvím aplikačního serveru.

8 Výpočetní úlohy

Tato komponenta se zabývá řešením vybraných typů úloh. Pro implementační účely byla vybrána pouze úloha ustáleného chodu ES, nicméně je možné okruh těchto úloh dále rozšiřovat. Výpočty realizuje níže popsaný výpočetní uzel, který implementuje vybrané typy úloh a s nimi související metody výpočtů. Případně by mohly být úlohy k provedení předávány na specializovaný cluster určený k výpočtům. To je však v tomto okamžiku pouze námět k úvaze.

8.1 Výpočetní uzel

Výpočetní uzel je jedna konkrétní instance této komponenty. Jeho úkolem je zpracovat úlohy, které má ve svojí frontě. Tato fronta je reprezentována úlohami k němu přiřazenými. Ty jsou uloženy v databázi a přístupné pomocí aplikačního serveru. Je vytvořen jako tzv. *daemon*, čímž se označuje služba běžící na pozadí. Z hlediska funkčnosti uzel implementuje všechny dostupné metody pro řešení problémů, které jsou nabízené k provádění výpočtu úloh.

V současném stavu lze zřizovat více jeho instancí, čímž se myslí zejména jeho spuštění na více serverech. Z hlediska výkonu by nebylo optimální jeho vícenásobné spuštění na jednom serveru z toho důvodu, že by se díky více jeho instancím zvýšil čas potřebný ke zpracování úlohy. To je samozřejmě nutné uvažovat vzhledem k počtu dostupných jader daného serveru. Ideálně by na serveru mohlo být spuštěno tolik jeho instancí, kolik má jader. Tím by se jednotlivé úlohy vzájemně neomezovaly. Nejedná se v tomto případě o situaci, kdy by se jednotlivé instance vzájemně blokovaly, ale pouze si navzájem ubíraly výpočetní výkon. Jednotlivé instance mezi sebou nijak nekomunikují.

Tato komponenta je vytvořena v jazyce Java a využívá konektor na aplikační server, který je blíže popsán v kapitole 5.7. Stejně jako komponenta pro import dat, tak i tato ze stejných důvodů využívá pro potřeby autorizace autorizační proces *pověření uživatele*. Jediný rozdíl spočívá ve faktu, že *účelový přístupový účet* k tomu použitý je vázán na systémového uživatele a nikoli na konkrétního uživatele. To je z důvodu, že samotné provedení výpočtů neprovádí konkrétní uživatel, nýbrž tato komponenta, či-li automatizovaný program. A proto jsou všechny takto vytvořené *účelové přístupové účty* svázány s identitou systémového uživatele.

Prostřednictvím komponenty editoru lze do fronty úloh k výpočetnímu uzlu přiřazovat nové úlohy či upravovat stávající (viz příloha F).

8.2 Životní cyklus úlohy

Úloha je jedna procesní jednotka při vykonávání výpočtů. Tato jednotka obsahuje údaje o jejím tvůrci, volbě výpočetní metody, preferovaný výpočetní uzel a schema, nad kterým se má zvolená metoda spustit. Úloha se může nacházet v jednom z následujících stavů:

- **připravující se** – úlohu uživatel ještě připravuje a může dojít k její modifikaci,
- **potvrzená k výpočtu** – úlohu již uživatel nemůže modifikovat a je připravena k předání výpočetnímu uzlu, až bude mít volné kapacity,
- **provádějící se** – výpočetní uzel provádí její výpočet a ještě nesignalizoval dokončení,
- **dokončena** – výpočetní uzel dokončil výpočet, navrátil vypočtené hodnoty či chybový stav.

8.3 Ustálený chod ES

Ustálený chod ES je přenosový stav soustavy, při kterém je možné považovat její proměnné parametry za konstantní. Hlavním důvodem, proč se provádí výpočet ustáleného chodu ES, je zjišťování činných a jalových výkonů, napěťových poměrů v jednotlivých prvcích a uzlech ES, ztrát, či zjišťování, zda není nějaké vedení proudově přetíženo. Všechny tyto informace jsou nutné k řízení provozu a navrhování dalšího rozvoje soustavy samé. Dále se hodnoty vypočítané při ustáleném stavu používají při řešení řady optimalizačních úloh jako je hospodárné rozdělování výroby činných a jalových výkonů, optimální regulace napětí a hodnocení spolehlivosti ES. Výpočet se běžně provádí pro maximální a minimální zatížení sítě a při výpočtu uvažujeme, že soustava (zdroje, přenosové prvky a odběry) je souměrná, což znamená, že úlohu můžeme řešit jako jednofázovou síť. Všechny prvky ES (vedení, transformátory, generátory, zátěže) se při výpočtu ustáleného chodu nahrazují pomocí jejich náhradních schemat. [1]

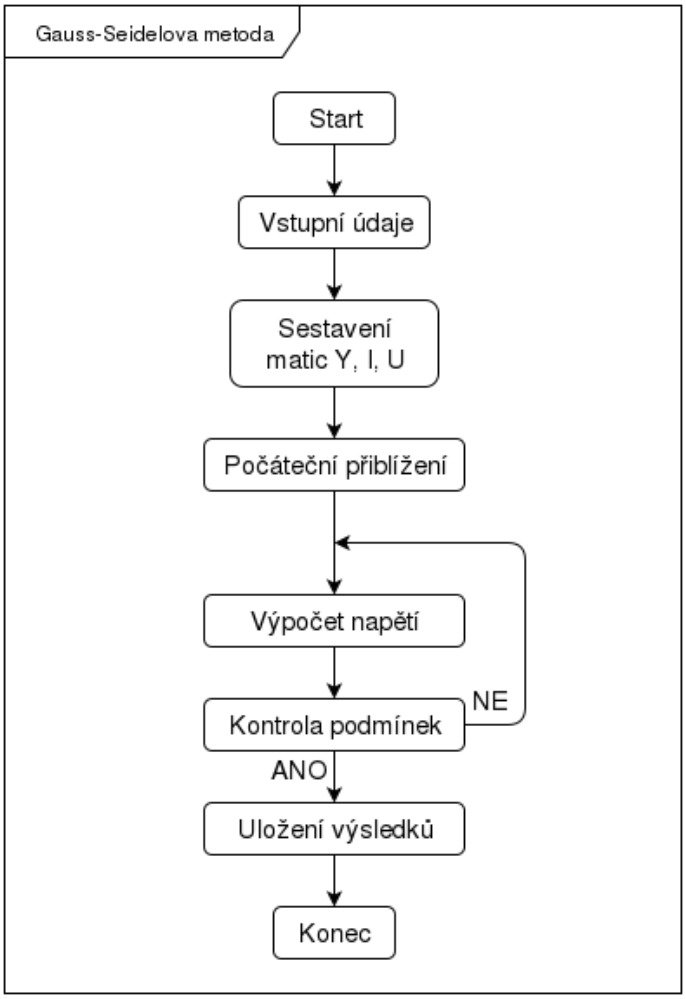
Při řešení ustáleného chodu ES se využívají matematické numerické metody, které pomocí iteračních cyklů naleznou řešení. Existuje více numerických metod pro tyto účely použitelných, kdy mezi nejpoužívanější patří Gauss-Seidelova metoda a Newton-Raphsonova metoda. V následujících kapitolách jsou tyto metody popsány. Každá z nich obsahuje své výhody i nevýhody.

8.3.1 Gauss-Seidelova metoda

Gauss-Seidelova metoda je z hlediska matematického zápisu jedna z nejjednodušších. To zároveň zapříčiňuje vysoký počet nutných iterací. Zejména u sítí s řádově stovkami až tisíci uzly se počet iterací pohybuje velmi vysoko. Hlavní pozorované parametry při řešení jsou velikost a úhel napětí. Následující výpočty probíhají převážně

v komplexních číslech, což je také jeden z rozdílů mezi touto metodou a Newton-Raphsonovou metodou, jenž je uvedena v další kapitole.

Výpočet probíhá v několika krocích. Tento postup je znázorněn na obrázku 8.1. Nejprve jsou načtena data daného schematu sítě z aplikačního serveru. Následně je sestavena admitanční matice Y, matice proudu I a matice napětí U. Admitanční matice je typu $n \times n$, kde n vyjadřuje počet uzlových bodů sítě. Matice proudu a napětí jsou obě sloupcové matice také o velikosti n. Do matice napětí se nejprve doplní známé hodnoty napětí uzlových bodů. Následně se při počátečním přiblížení provede doplnění neznámých hodnot napětí u ostatních bodů. Doplňuje se hodnotou hladinového napětí bodu. Mezi uzlové body se známým napětím se řadí napájecí body a generátory. Všechny ostatní typy bodů mají toto napětí zpočátku neznámé. Sestavení admitanční matice je založeno na spočtení vhodného náhradního schematu pro jednotlivé úseky mezi uzlovými body a je popsáno v [28].



Obrázek 8.1: Postup výpočtu G-S metodou

Samotný výpočet jedné iterace (viz obrázek 8.1) je založen na výpočtu napětí pro každý uzlový bod, který měl zpočátku sestavování matice napětí toto napětí

neznámé. U uzlových bodů se známým napětím je jejich hodnota konstantní po celou dobu výpočtu. U ostatních je každou iterací počítána a zpřesňována. Toto je popsáno rovnicemi v [27]. Po každé iteraci se kontroluje, zda-li je rozdíl vypočteného napětí v aktuální iteraci a napětí v předešlé iteraci v povoleném rozsahu přesnosti. Pokud tomu tak je, iterace se ukončí a proběhne pouze uložení výsledků zpět na aplikační server. V opačném případě se vypočtené napětí uloží do matice napětí a přepíše tak jeho aktuální hodnotu. Následují další iterace, které jsou prováděny do okamžiku dosažení požadované přesnosti. Je však omezen jejich maximální počet. Po ukončení výpočtu se na aplikační server uloží výsledky platné po poslední iteraci. V případě, kdy je přesažen limit maximálního počtu iterací, je tato skutečnost také uložena jako výsledek provedené úlohy.

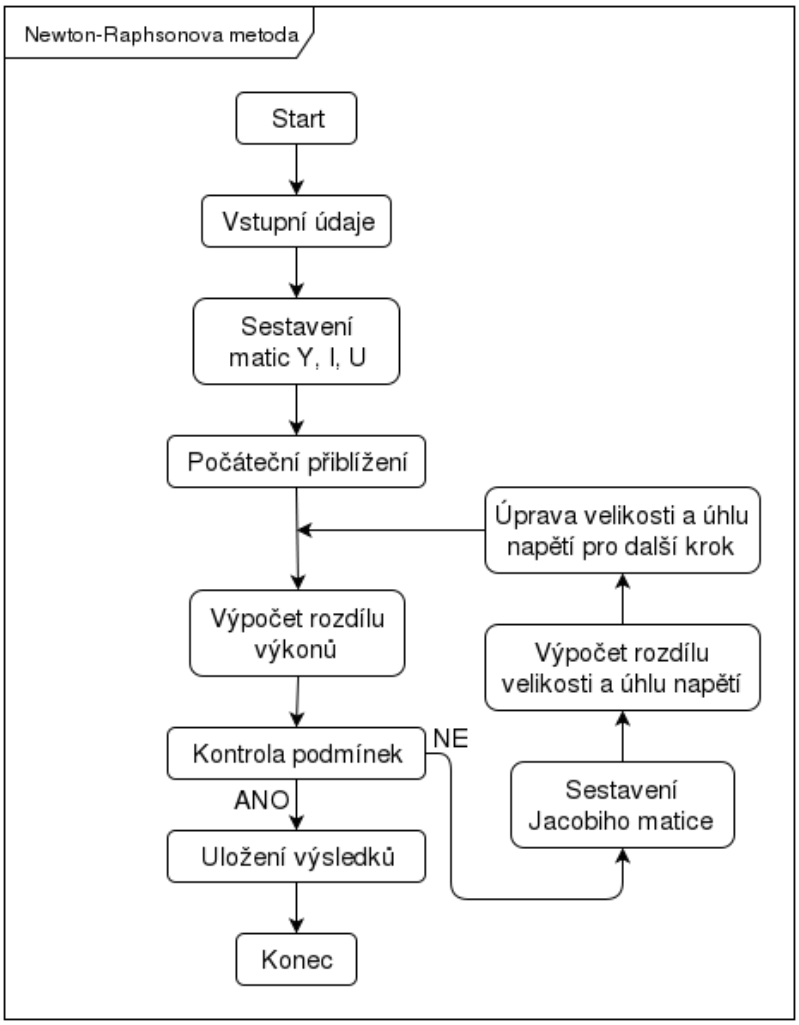
Gauss-Seidlova metoda se vyznačuje oproti jiným iteračním metodám poměrně krátkou dobou výpočtu na jeden iterační krok. Nevýhodou je poměrně pomalá konvergence. Je potřeba volit vysokou přesnost výpočtu, jinak je možné, že nepřesně vypočítaná napětí způsobí chybu v tocích výkonů a uzlových bilancích výkonů. [27]

8.3.2 Newton-Raphsonova metoda

Newton-Raphsonova metoda využívá metodu tečen, pomocí které se velice rychle blíží ke hledanému řešení. Tato metoda je aplikována na matematickém algoritmu, který rapidně sníží počet iterací, a proto se tato metoda užívá u rozsáhlejších sítí pro jejich rychlé vyřešení. Bohužel tato metoda není z hlediska matematické stability tak spolehlivá jako Gauss-Seidelova metoda. Při špatném zadání vstupních hodnot může metoda divergovat a nedosáhnout tak konečného řešení nebo konverguje k jinému řešení. Naopak výhodou této metody je, že v algoritmu se počítá pouze s reálnými čísly. [27]

Algoritmus výpočtu je zachycen na obrázku 8.2. První část algoritmu je shodná s předešlou metodou. Jedná se o spuštění algoritmu, načtení dat vybraného schematu sítě z aplikačního serveru a následné sestavení admitanční matice a matic proudů a napětí. Právě z důvodu shodnosti tohoto postupu se další obsah zaměřuje na vlastní výpočet jednotlivé iterace.

V každé iteraci se počítá rozdíl aktuálně vypočtených činných a jalových výkonů od jejich předem zadaných hodnot. Tyto hodnoty činných a jalových výkonů jsou konstantní pro všechny iterace. Pokud je rozdíl výkonů v požadovaném rozsahu přesnosti, iterace se ukončí a proběhne uložení vypočtených parametrů, především hodnoty velikosti a úhlu napětí v uzlových bodech. V opačném případě je sestavena Jacobiho matice (viz [27]), která je nezbytná pro určení rozdílu napětí. Rozdíl napětí se počítá z vypočteného rozdílu výkonů a Jacobiho matice. Poté jsou o zjištěný rozdíl upravena napětí v jednotlivých uzlových bodech pro následující iteraci. Další iterace se provádějí do okamžiku dosažení požadované přesnosti. Je zde omezen jejich maximální počet. Po ukončení výpočtu se na aplikační server uloží výsledky platné po poslední iteraci. V případě, kdy je přesažen limit maximálního počtu iterací, je tato skutečnost také uložena jako výsledek provedené úlohy.



Obrázek 8.2: Postup výpočtu N-R metodou

8.4 Testování

Testování této komponenty probíhalo za pomocí následujících technik. Po vytvoření daného kódu byl kód testován programátorem. Vzhledem k faktu, že se zde využíval i API Connector z kapitoly 5.7, který byl samostatně testován, tak tato část nebyla zvláště testována. Nad celou komponentou pak proběhlo systémové testování, jenž mělo za úkol ověřit její funkčnost. To bylo provedeno autorem. Tímto testováním komponenta prošla.

Faktická správnost vypočtených výsledků byla ověřena porovnáním hodnot. Pro tyto účely bylo vytvořeno testovací schema ES, které bylo manuálně vypočteno. Zjištěné hodnoty byly porovnány s hodnotami vypočtenými komponentou výpočtů. Důvodem byla skutečnost, že v aktuální konfiguraci systému RIS nebyla u konzultanta korektně nastavena možnost výpočtu sítí hladiny VN a NN, a tudíž nebylo možné ověření provést vůči převzatým schematům Želkovice a Dobruška.

9 Závěr

Celou tuto práci je možné charakterizovat jedním slovem – platforma. Platforma určená pro správu a vizualizaci elektrizační soustavy a k vědeckým výpočtům nad zvolenou soustavou. Platforma, která je propojitelná s externími systémy a která je volně dostupná pod svobodnou licencí. Platforma, jejíž funkčnost byla ověřena u konzultanta a lze ji dále rozvíjet.

Hlavním cílem této práce byl způsob návrhu platformy a návrh jejích programů a služeb z pohledu softwarového architekta. To vše s ohledem na aktuální webové technologie. Byla to jedna z ústředních otázek, zda-li, jak a do jaké míry je možné webové technologie zakomponovat do platformy. V jejím rámci byly identifikovány komponenty, jejichž návrh byl na počátku práce diskutován a posléze implementován. Jádrem platformy, a též hlavní klíčovou komponentou, je aplikační server. Ten je navržen jako server poskytující webové služby prostřednictvím svého aplikačního programového rozhraní. Ostatní komponenty, s výjimkou poskytovatele identit, jeho prostřednictvím přistupují a pracují s daty uloženými v databázi. Mezi vyvinuté komponenty komunikující s aplikačním serverem patří webový editor, který poskytuje grafické rozhraní uživateli, komponenta pro import dat z externího systému a výpočetní uzel řešící problematiku výpočtů nad schematem elektrizační sítě.

Pro praktické účely implementace výpočetního uzlu proběhlo seznámení s úlohou řešící ustálený chod elektrizační sítě. V tomto rámci byly nastudovány dvě metody toto řešící, konkrétně Gauss-Seidelova a Newton-Raphsonova. Obě byly implementovány a dle zjištěných poznatků porovnáním s očekávanými výsledky označeny za vyhovující.

Se svolením konzultanta byla z jeho externího systému převzata dvě schemata sítí z obce Želkovice a města Dobrušky. Tato schemata byla importována do platformy a byla korektně zobrazena v prostředí komponenty editoru, kde s nimi bylo možné provádět dostupné operace.

Lze konstatovat, že cíle vytyčené v zadání byly splněny. Platformu se podařilo navrhnout za výrazného použití webových technologií s přihlédnutím k výkonovým potřebám jednotlivých komponent. Jednotlivé komponenty byly průběžně testovány a těmito testy prošly.

Platformu je možné dále rozšiřovat. A to ať cestou přidávání jednotlivých typů úloh řešených nad schematem sítě, tak z pohledu použitých technologií. To lze demonstrovat na rozšíření funkcí webového editoru či vzniku grafického rozhraní pro správu identit uživatelů v rámci komponenty poskytovatele identit. Přínosem platformy je její otevřenost, dostupnost pod svobodnou licencí, rozšiřitelnost a snadná modifikovatelnost.

Literatura

[1] BUREŠ, Martin. *Počítačová reprezentace elektrické rozvodné sítě*. Liberec, 2015. Magisterský projekt.

[2] Řídící a informační systém RIS - SCADA/EMS. *ELEKTROSYSTEM, a.s.* [online]. Brno: ELEKTROSYSTEM, 2014 [cit. 2016-04-01]. Dostupné z: http://www.esys.cz/ris.php

[3] Software as a service. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2014 [cit. 2016-04-01]. Dostupné z: https://cs.wikipedia.org/wiki/Software_as_a_service

[4] PATTON, Ron. *Testování softwaru*. Praha: Computer Press, 2002. Programování. ISBN 80-722-6636-5.

[5] BUCHALCEVOVÁ, Alena. *Metodiky budování informačních systémů*. Praha: Oeconomica, 2009. ISBN 978-80-245-1540-3.

[6] DŽAFERAGIĆ, Amir. *Agilní přístup k řízení softwarových projektů*. Brno, 2011. Diplomová práce. MASARYKOVA UNIVERZITA.

[7] Agilní vývoj: Scrum. *Zdrojak.cz* [online]. Praha: Devel.cz Labs, 2009 [cit. 2016-04-01]. Dostupné z: https://www.zdrojak.cz/clanky/agilni-vyvoj-scrum/

[8] BECK, Kent. *Extrémní programování*. 1. vyd. Praha: Grada, 2002. Moderní programování. ISBN 80-247-0300-9.

[9] Metodika Testování — Testování softwaru. *Testování softwaru* [online]. Praha: Testování softwaru, 2011 [cit. 2016-04-01]. Dostupné z: http://testovanisoftwaru.cz/category/metodika-testovani/

[10] Mpi77 — GitHub. *GitHub.com* [online]. 2016 [cit. 2016-04-01]. Dostupné z: https://github.com/mpi77

[11] RFC 6749 - The OAuth 2.0 Authorization Framework. *The Internet Engineering Task Force* [online]. California, USA: The Internet Engineering Task Force, 2012 [cit. 2016-04-01]. Dostupné z: https://tools.ietf.org/html/rfc6749

[12] OAuth 2.0 Server PHP. *GitHub.com* [online]. San Francisco, 2016 [cit. 2016-04-01]. Dostupné z: http://bshaffer.github.io/oauth2-server-php-docs/

[13] BOOTH, David, et al. W3C [online]. 11. 2. 2004 [cit. 2016-04-01]. Web Services Architecture. Dostupné z: http://www.w3.org/TR/ws-arch/

[14] KADLEC, Jiří. *REST a webové služby v jazyce Java*. Brno, 2010. Diplomová práce. MASARYKOVA UNIVERZITA.

[15] RICHARDSON, Leonard a Sam RUBY. *RESTful web services*. Farnham: O'Reilly, 2007. ISBN 05-965-2926-0.

[16] API Blueprint Specification. *API Blueprint* [online]. 2016 [cit. 2016-04-01]. Dostupné z: https://apiblueprint.org/documentation/specification.html

[17] Specification - Swagger. *Swagger* [online]. 2016 [cit. 2016-04-01]. Dostupné z: http://swagger.io/specification/

[18] RAML Version 0.8 Specification. *GitHub.com* [online]. 2016 [cit. 2016-04-01]. Dostupné z: https://github.com/raml-org/raml-spec/blob/master/raml-0.8.md

[19] Phalcon 2.0.10 Documentation. *Phalcon* [online]. 2016 [cit. 2016-04-01]. Dostupné z: https://docs.phalconphp.com/en/latest/index.html

[20] Cross-site Scripting (XSS). *OWASP* [online]. 2016 [cit. 2016-04-01]. Dostupné z: https://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29

[21] SQL Injection. *OWASP* [online]. 2016 [cit. 2016-04-01]. Dostupné z: https://www.owasp.org/index.php/SQL_Injection

[22] Cross-Site Request Forgery (CSRF). *OWASP* [online]. 2016 [cit. 2016-04-01]. Dostupné z: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29

[23] RFC 6797 – HTTP Strict Transport Security (HSTS). *The Internet Engineering Task Force* [online]. California, USA: The Internet Engineering Task Force, 2012 [cit. 2016-04-01]. Dostupné z: https://tools.ietf.org/html/rfc6797

[24] RFC 7540 – Hypertext Transfer Protocol Version 2 (HTTP/2). *The Internet Engineering Task Force* [online]. California, USA: The Internet Engineering Task Force, 2015 [cit. 2016-04-01]. Dostupné z: https://tools.ietf.org/html/rfc7540

[25] Cross-Origin Resource Sharing. *World Wide Web Consortium (W3C)* [online]. World Wide Web Consortium (W3C), 2014 [cit. 2016-04-01]. Dostupné z: https://www.w3.org/TR/cors/

[26] AngularJS Developer Guide. *AngularJS* [online]. 2016 [cit. 2016-04-01]. Dostupné z: https://docs.angularjs.org/guide

[27] TAUŠ, Vladimír. *Výpočet ustáleného chodu sítě 110kV*. Brno, 2008. Diplomová práce. VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ.

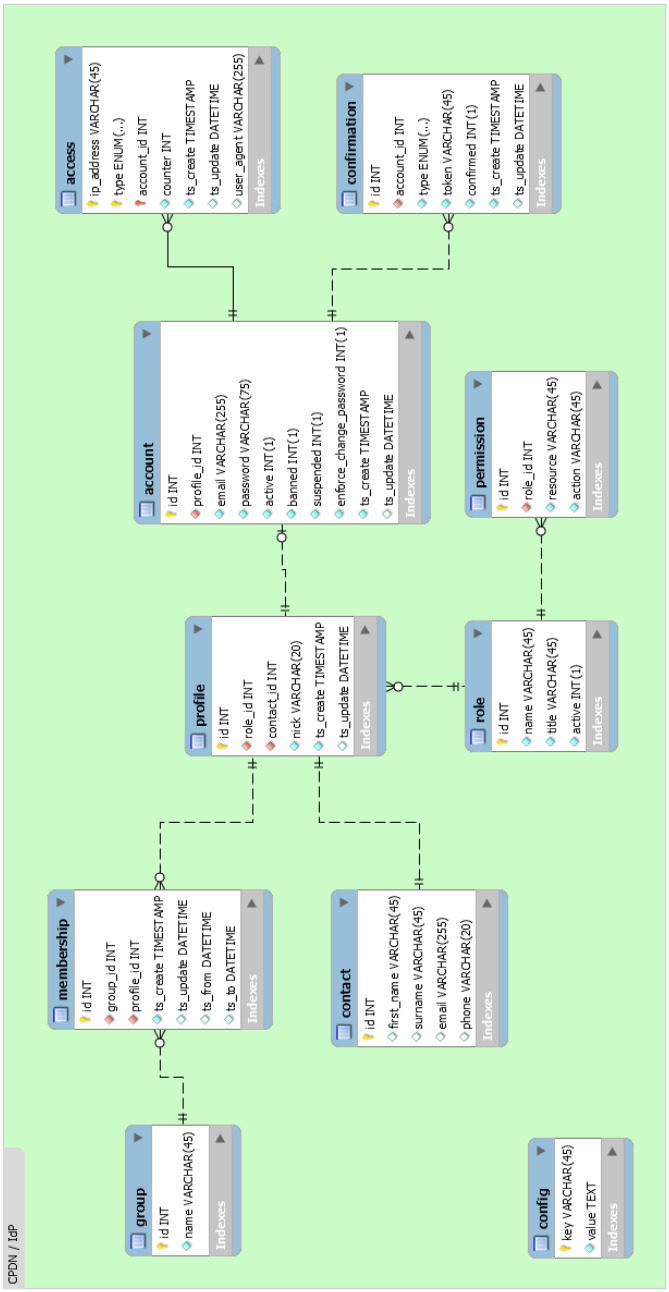
[28] ČELEDA, Jiří. *Řešení napěťové stability elektrizačních soustav v ustáleném stavu*. Plzeň, 2013. Diplomová práce. ZÁPADOČESKÁ UNIVERZITA V PLZNI.

Obsah CD

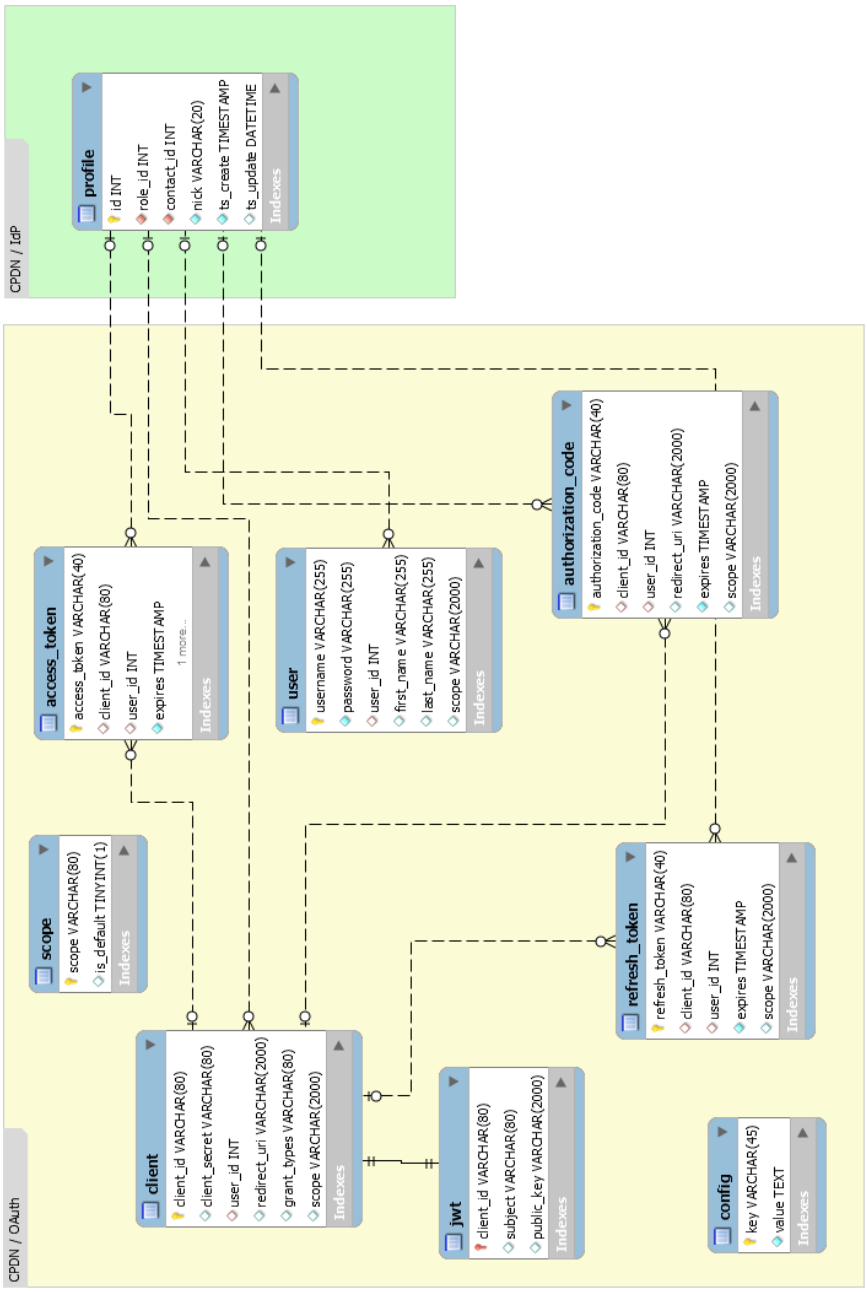
- **/soustavy/** – převzaté ES ze systému RIS
- **/2016-BURES-MARTIN.tex** – zdrojový soubor zprávy diplomové práce
- **/2016-BURES-MARTIN.pdf** – pdf verze zprávy diplomové práce

Přílohy

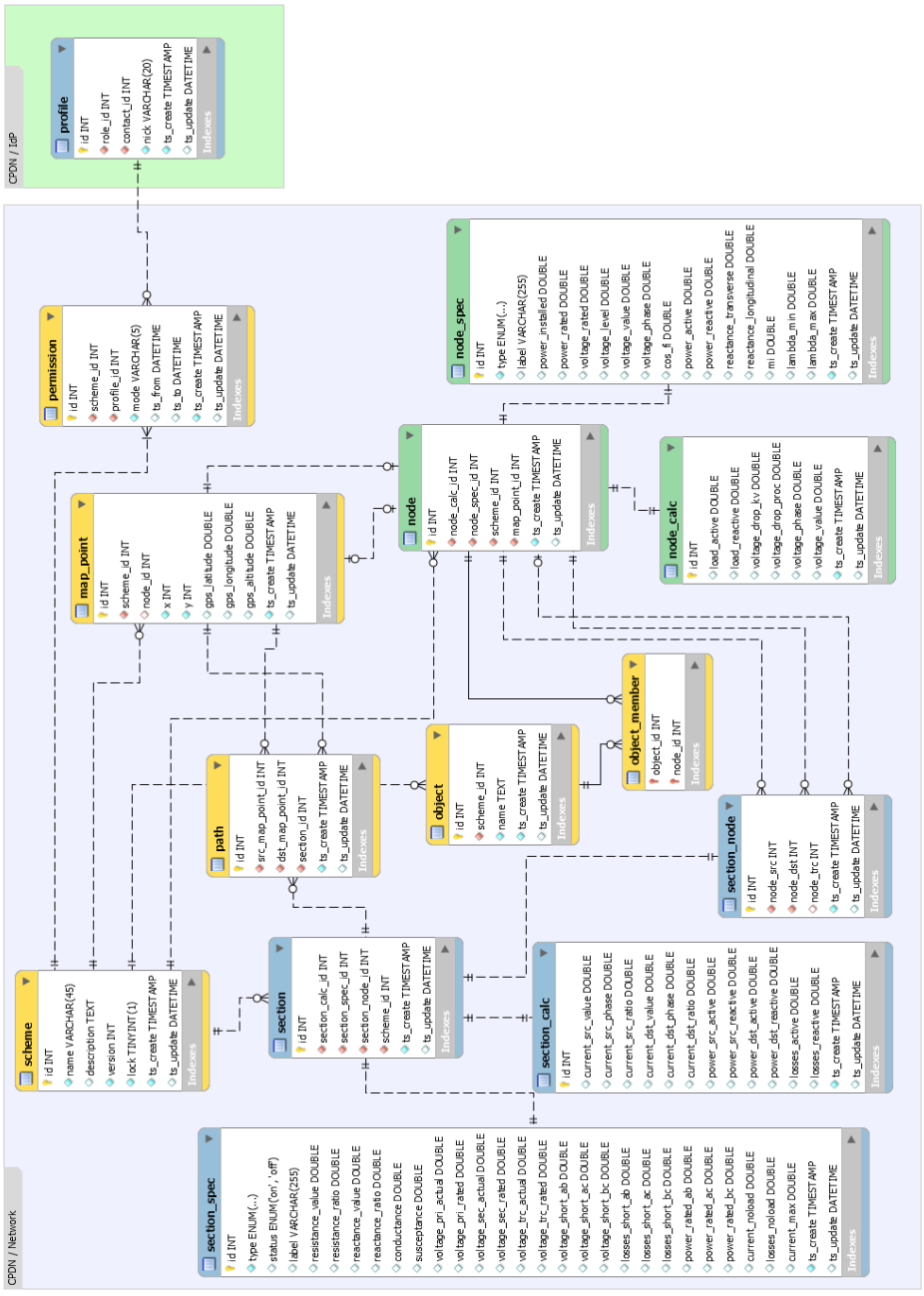
A ER model databáze Poskytovatel identit



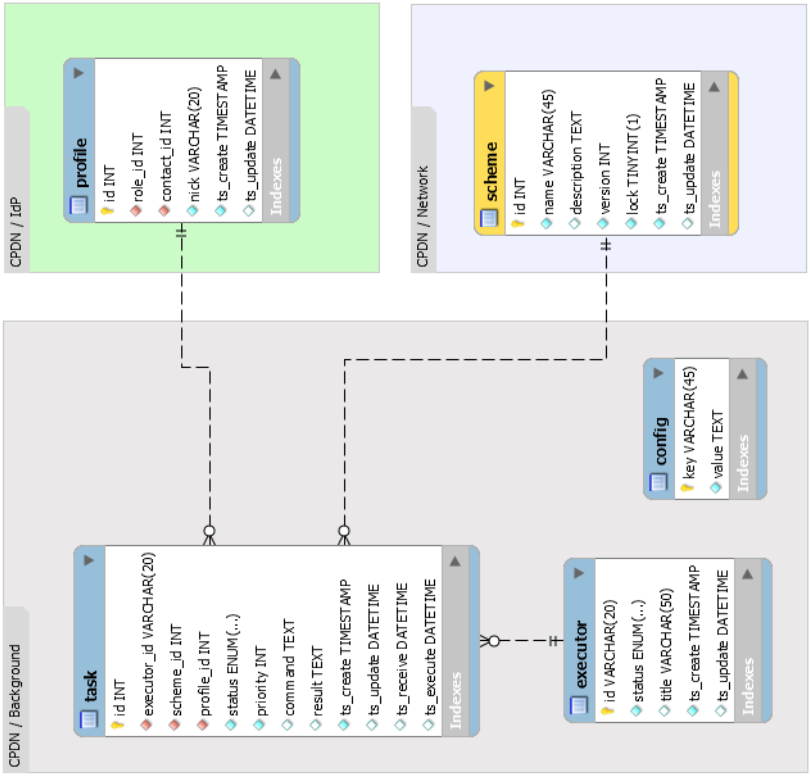
B ER model databáze OAuth



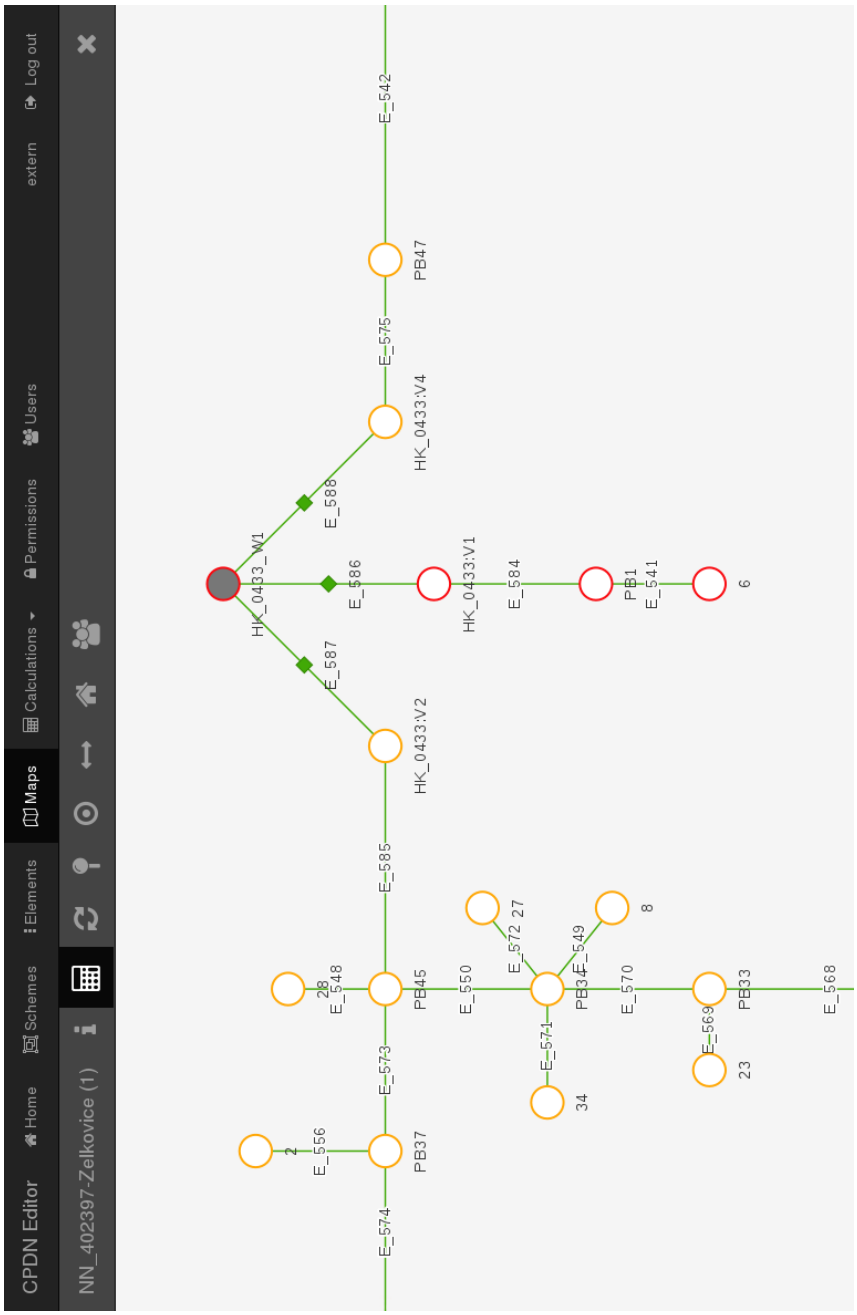
C ER model databáze Síť



D ER model databáze Výpočty



E Editor – schema sítě Želkovice



F Editor – ovládání úloh